Teach your (micro)services speak Protocol Buffers with gRPC.

Mihai Iachimovschi

@mishunika
✉ mihai.iachimovschi@gmail.com

What’s inside?

What’s inside?

–Message serialization and deserialization

What’s inside?

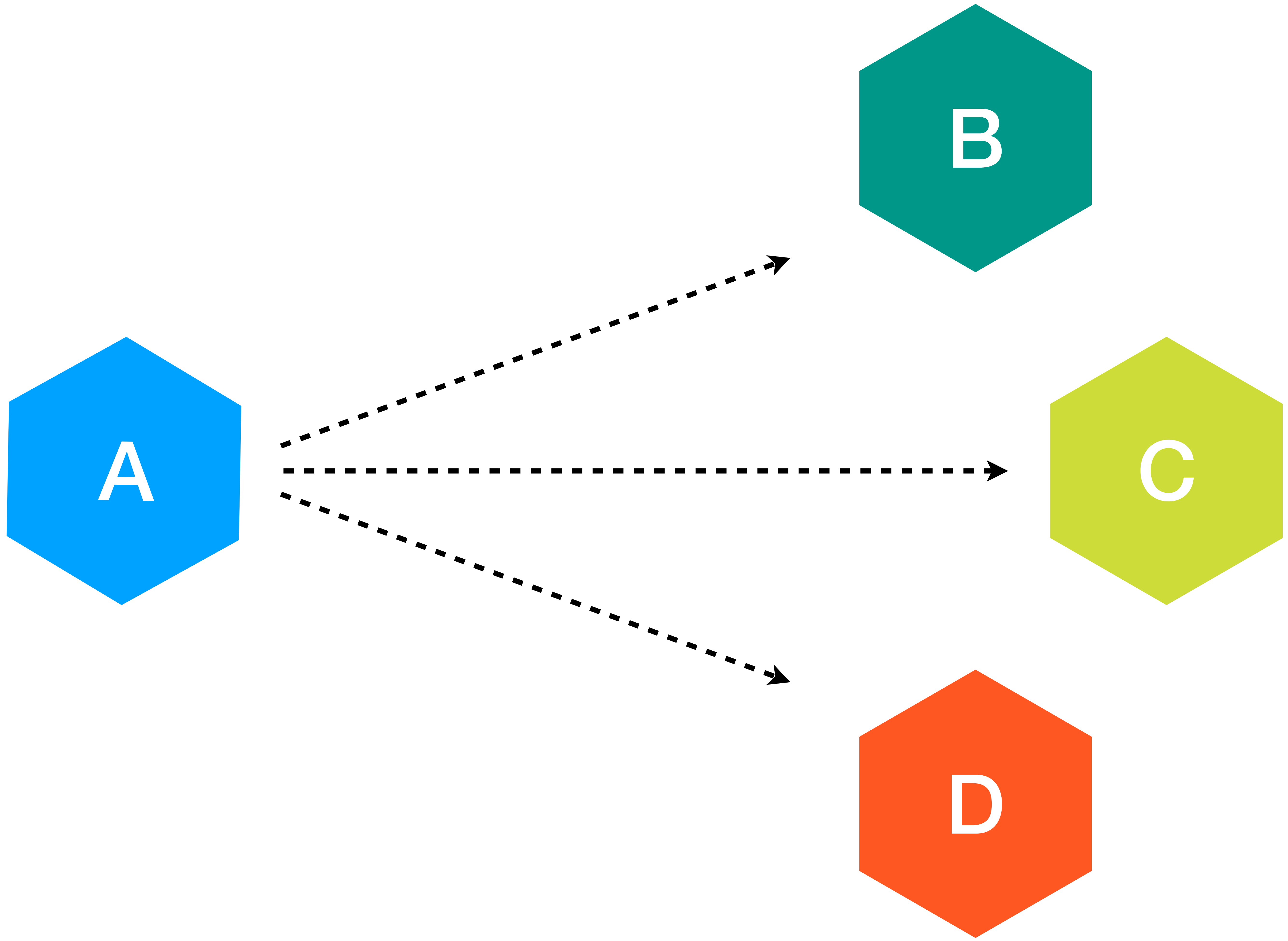
–Message serialization and deserialization

–Message transport

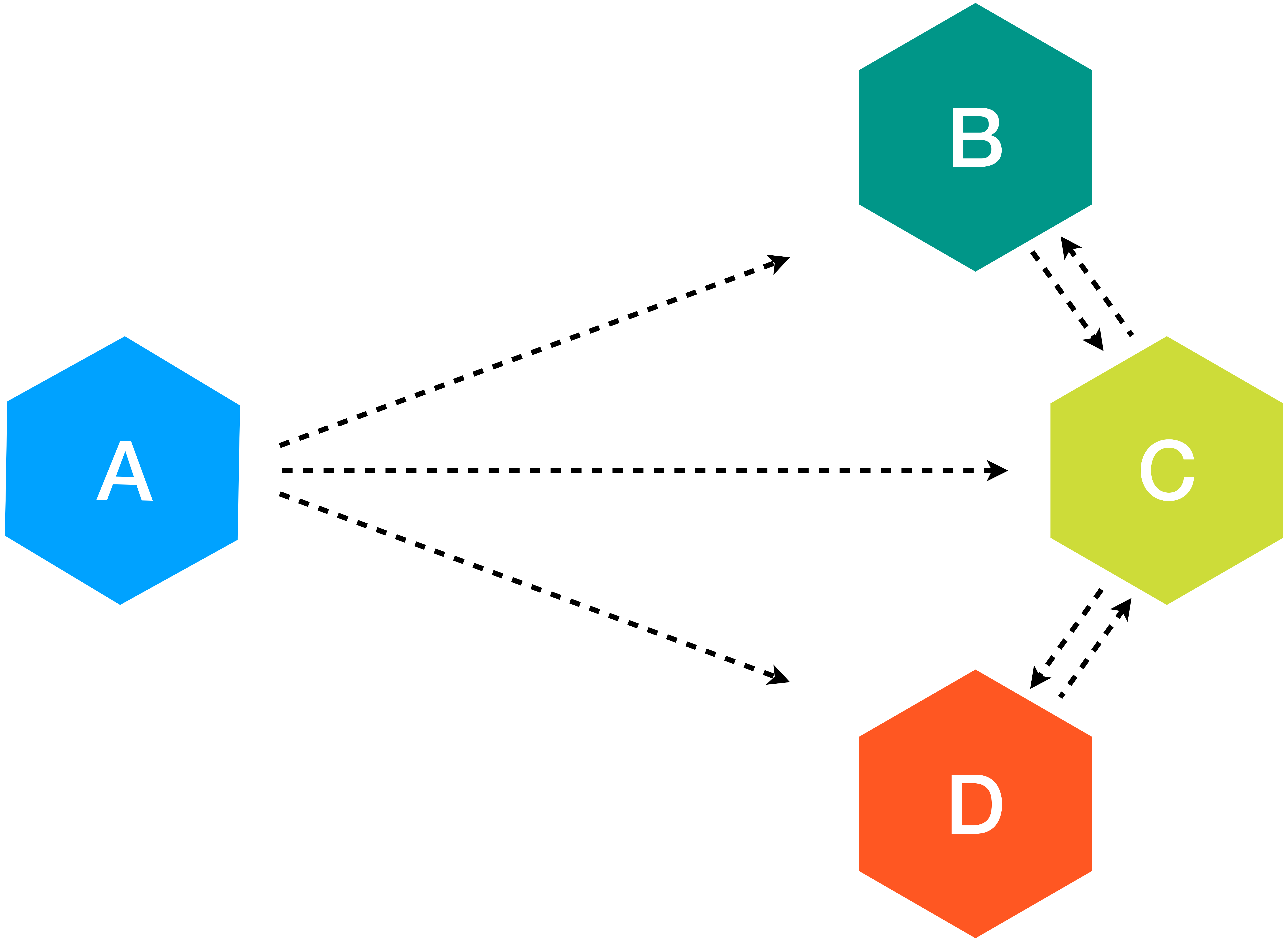
What’s inside?

- Message serialization and deserialization
- Message transport
- Services diversity

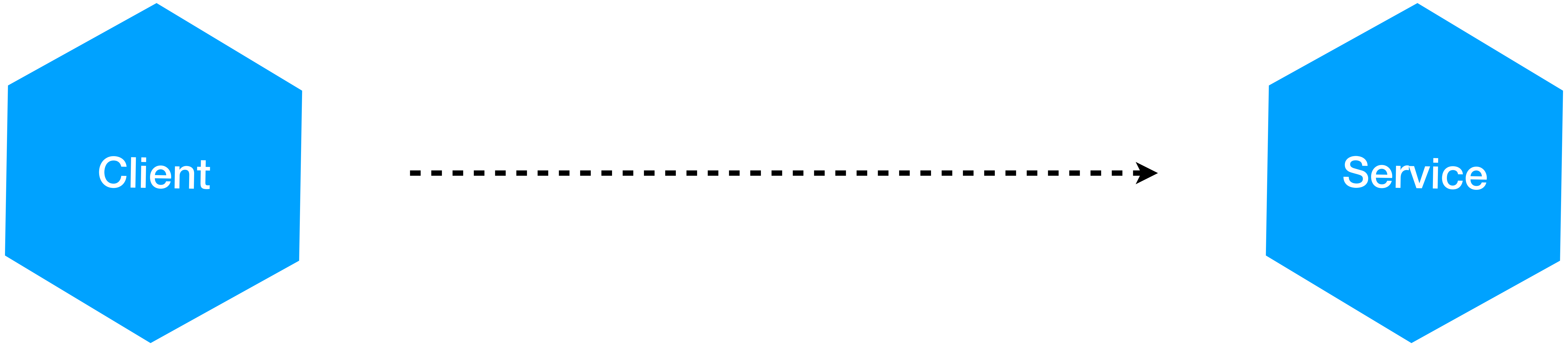
How it works in real life



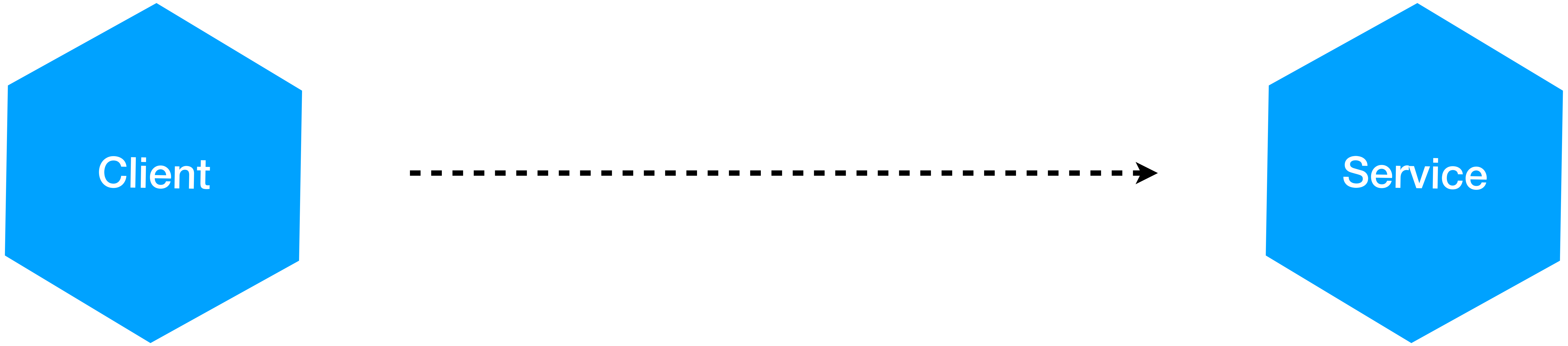
How it works in real life



In a nutshell



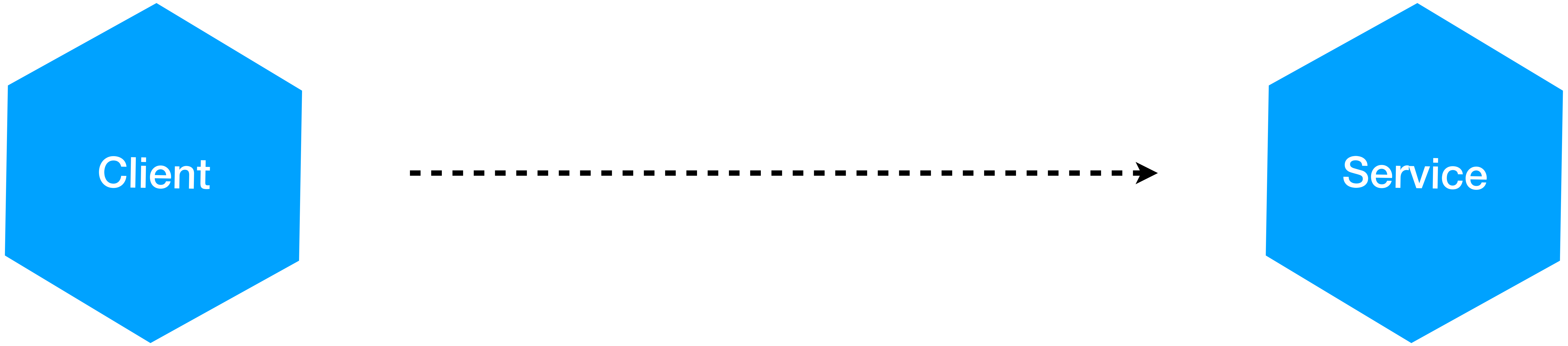
In a nutshell



–Over HTTP

–Serialized to JSON

In a nutshell



–Over HTTP

–Serialized to JSON

–Proprietary protocol

–Remote objects

JSON advantages

JSON advantages

–Human readable

JSON advantages

–Human readable

–Schema-less

JSON advantages

–Human readable

–Schema-less

–Language agnostic

JSON disadvantages

JSON disadvantages

–Human readable

JSON disadvantages

–Human readable Isn’t it a benefit?

JSON disadvantages

–Human readable Isn’t it a benefit?

–Schema-less

JSON disadvantages

– Human readable   Isn't it a benefit?

– Schema-less   Isn't it a benefit as well?

JSON disadvantages

–Human readable Isn't it a benefit?

–Schema-less Isn't it a benefit as well?

–Type-less

Protocol Buffers?

“Protocol buffers are Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data – think XML, but smaller, faster, and simpler.”

— https://developers.google.com/protocol-buffers/

Protocol Buffers?

“Protocol buffers are Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data – think ~~XML~~, but smaller, faster, and simpler.”

— https://developers.google.com/protocol-buffers/

Protocol Buffers?

"Protocol buffers are Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data – think ~~XML~~ JSON, but smaller, faster, and simpler."

— https://developers.google.com/protocol-buffers/

Protocol Buffers example

```
message Person {
  string name = 1;
  int32 id = 2;
  repeated string aliases = 3;
}
```

Protocol Buffers example

```
message Person {
   string name = 1;
   int32 id = 2;
   repeated string aliases = 3;
}
```

Protocol Buffers example

```
message Person {
  string name = 1;
  int32 id = 2;
  repeated string aliases = 3;
}
```

Protocol Buffers example

```
message Person {
   string name = 1;
   int32 id = 2;
   repeated string aliases = 3;
}
```

Protocol Buffers example

```
message Person {
  string name = 1;
  int32 id = 2;
  repeated string aliases = 3;
}
```

Protocol Buffers example

```
message Person {
  string name = 1;
  int32 id = 2;
  repeated string aliases = 3;
}
```

Protocol Buffers example

```
message Person {
  reserved 1, 2, 5;
  reserved "name";
  int32 id = 3;
  repeated string aliases = 4;
}
```

Why protocol buffers

Binary format

Why protocol buffers

Enforcing the schema

Why protocol buffers

Language neutral

Why protocol buffers

Out-of the box backward compatibility

Why protocol buffers

Out-of the box backward compatibility

Why protocol buffers

Out-of the box backward compatibility

```
if version == 3:
  ...
elif version > 4:
  if (version == 5):
    ...
  ...
```

Why protocol buffers

Generally faster

How to…

```
message Person {
  string name = 1;
  int32 id = 2;
  repeated string aliases = 3;
}
```

How to...

```
message Person {
  string name = 1;
  int32 id = 2;
  repeated string aliases = 3;
}
```

Proto message definition

How to...

```
message Person {
  string name = 1;
  int32 id = 2;
  repeated string aliases = 3;
}
```

Proto message definition

```
Person john = Person.newBuilder()
    .setId(1234)
    .setName("John Doe")
    .addAliases("ionel")
    .build();
```

How to...

```
message Person {
  string name = 1;
  int32 id = 2;
  repeated string aliases = 3;
}
```

Proto message definition

Object creation

```
Person john = Person.newBuilder()
    .setId(1234)
    .setName("John Doe")
    .addAliases("ionel")
    .build();
```

How to...

```
message Person {
  string name = 1;
  int32 id = 2;
  repeated string aliases = 3;
}
```

Proto message definition

Object creation

```
john = Person()
john.id = 1234
john.name = 'John Doe'
john.aliases.add(‘ionel’)
```

How to...

```
message Person {
  string name = 1;
  int32 id = 2;
  repeated string aliases = 3;
}
```

Proto message definition

Object creation

```
john = Person(id=1234,
              name='John Doe',
              aliases=['ionel'])
```

Communication (REST-ish)

1. URI: https://api.example.com/person/42
2. Make an HTTP GET Request
3. Receive a plaintext JSON
4. Parse it
5. …

Request

```
GET /person/42 HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
Connection: keep-alive
Host: api.example.com
...
```

Response headers

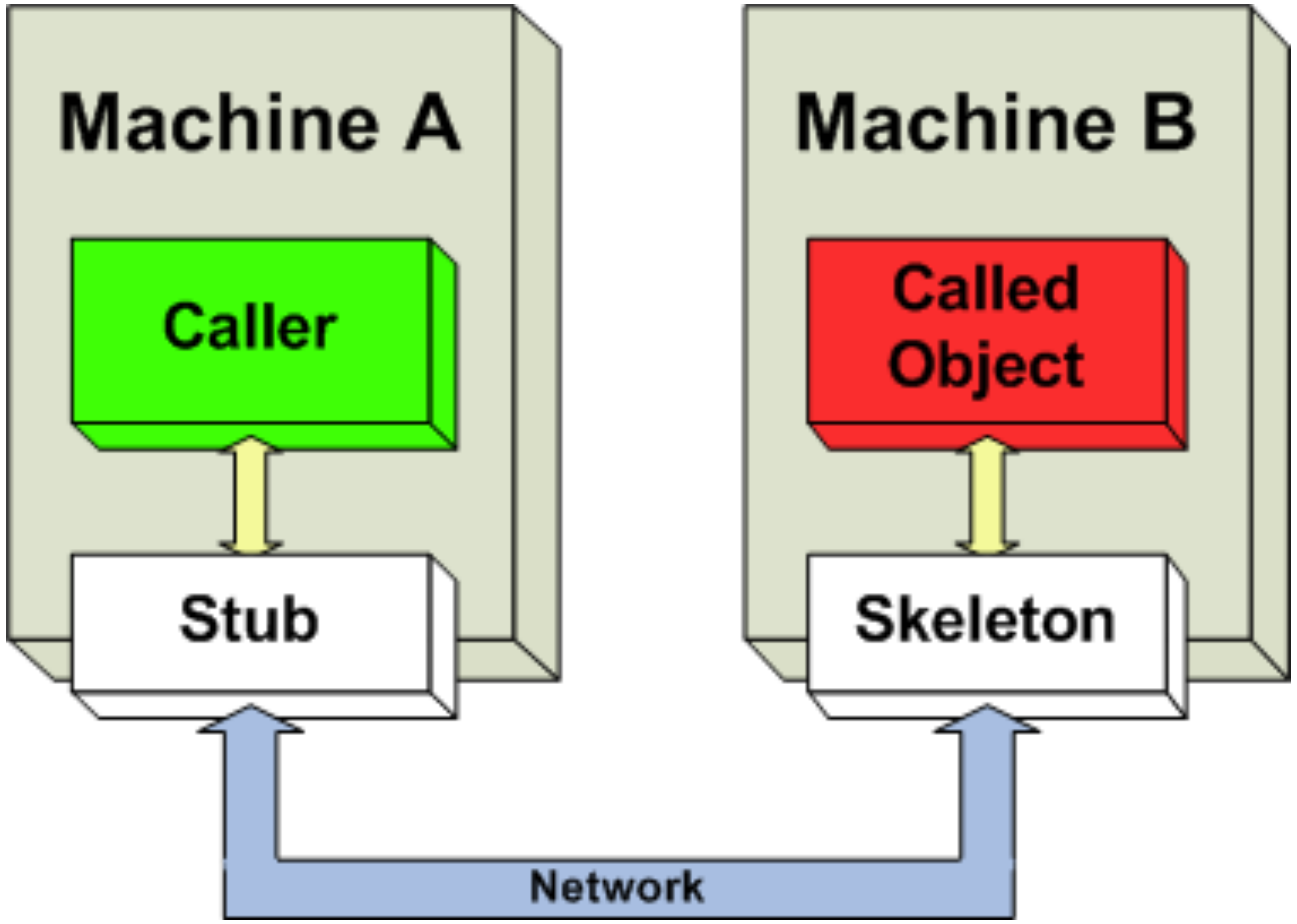
```
HTTP/1.1 200 OK
Access-Control-Allow-Credentials: true
Cache-Control: public, max-age=14400
Content-Encoding: gzip
Content-Type: application/json;
charset=utf-8
...
```

Response body

```
{
    "name": "John Doe",
    "id": 42,
    "aliases": [
        "ionel",
        "honzík"
    ]
}
```

https://www.youtube.com/watch?v=DtTKF5OcpsU

Distributed objects



https://en.wikipedia.org/wiki/Distributed_object

Distributed objects

“First Law of Distributed Object Design:
don't distribute your objects”.

— Martin Fowler

https://martinfowler.com/articles/distributed-objects-microservices.html

The 8 Fallacies of distributed computing

1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn't change.
6. There is one administrator.
7. Transport cost is zero.
8. The network is homogeneous.

http://www.rgoarchitects.com/Files/fallacies.pdf

Keep in mind…

> “Anything that can go wrong will go wrong.”
>
> —Murphy’s Law

So, what's next?

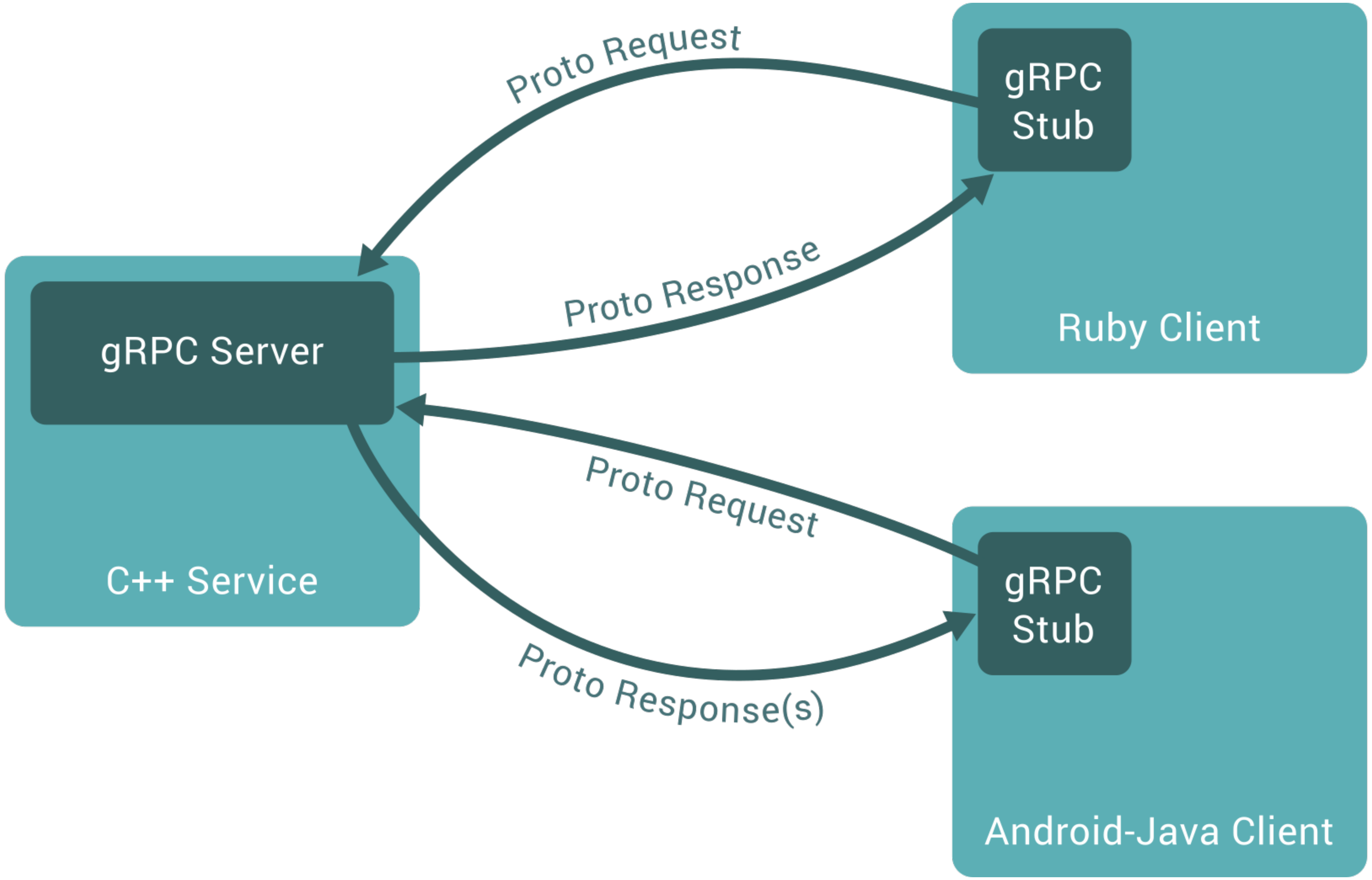
Services not Objects, Messages not References

http://www.grpc.io/blog/principles

Coverage & Simplicity

http://www.grpc.io/blog/principles

How does it work



Service definition

```
service RoutePlanner {
    rpc GetRoutes (GetRoutesRequest)
        returns (GetRoutesResponse) {}
}
```

Service definition

```
service RoutePlanner {
    rpc GetRoutes (GetRoutesRequest)
        returns (GetRoutesResponse) {}
}
```

Service definition

```
service RoutePlanner {
    rpc GetRoutes (GetRoutesRequest)
        returns (GetRoutesResponse) {}
}
```

Service definition

```
service RoutePlanner {
    rpc GetRoutes (GetRoutesRequest)
        returns (GetRoutesResponse) {}
}
```

Service definition

```
service RoutePlanner {
    rpc GetRoutes (GetRoutesRequest)
        returns (GetRoutesResponse) {}
}
```

Service definition

```
service RoutePlanner {
    rpc GetRoutes (GetRoutesRequest)
        returns (GetRoutesResponse) {}
}
```

Service definition

```
message GetRoutesRequest {
  Location origin = 1;
  Location destination = 2;
}

message GetRoutesResponse {
  repeated Route routes = 1;
}
```

Generate server code

```
$ python -m grpc_tools.protoc \
      --proto_path=../protos \
      --python_out=. \
      --grpc_python_out=. \
      ../protos/route_planner.proto
```

Generate server code

```
$ python -m grpc_tools.protoc \
      --proto_path=../protos \
      --python_out=. \
      --grpc_python_out=. \
      ../protos/route_planner.proto
```

Generate server code

```
$ python -m grpc_tools.protoc \
      --proto_path=../protos \
      --python_out=. \
      --grpc_python_out=. \
      ../protos/route_planner.proto
```

Generate server code

```
$ python -m grpc_tools.protoc \
      --proto_path=../protos \
      --python_out=. \
      --grpc_python_out=. \
      ../protos/route_planner.proto
```

Generate server code

```
$ python -m grpc_tools.protoc \
      --proto_path=../protos \
      --python_out=. \
      --grpc_python_out=. \
      ../protos/route_planner.proto
```

Generate server code

```
$ python -m grpc_tools.protoc \
      --proto_path=../protos \
      --python_out=. \
      --grpc_python_out=. \
      ../protos/route_planner.proto
```

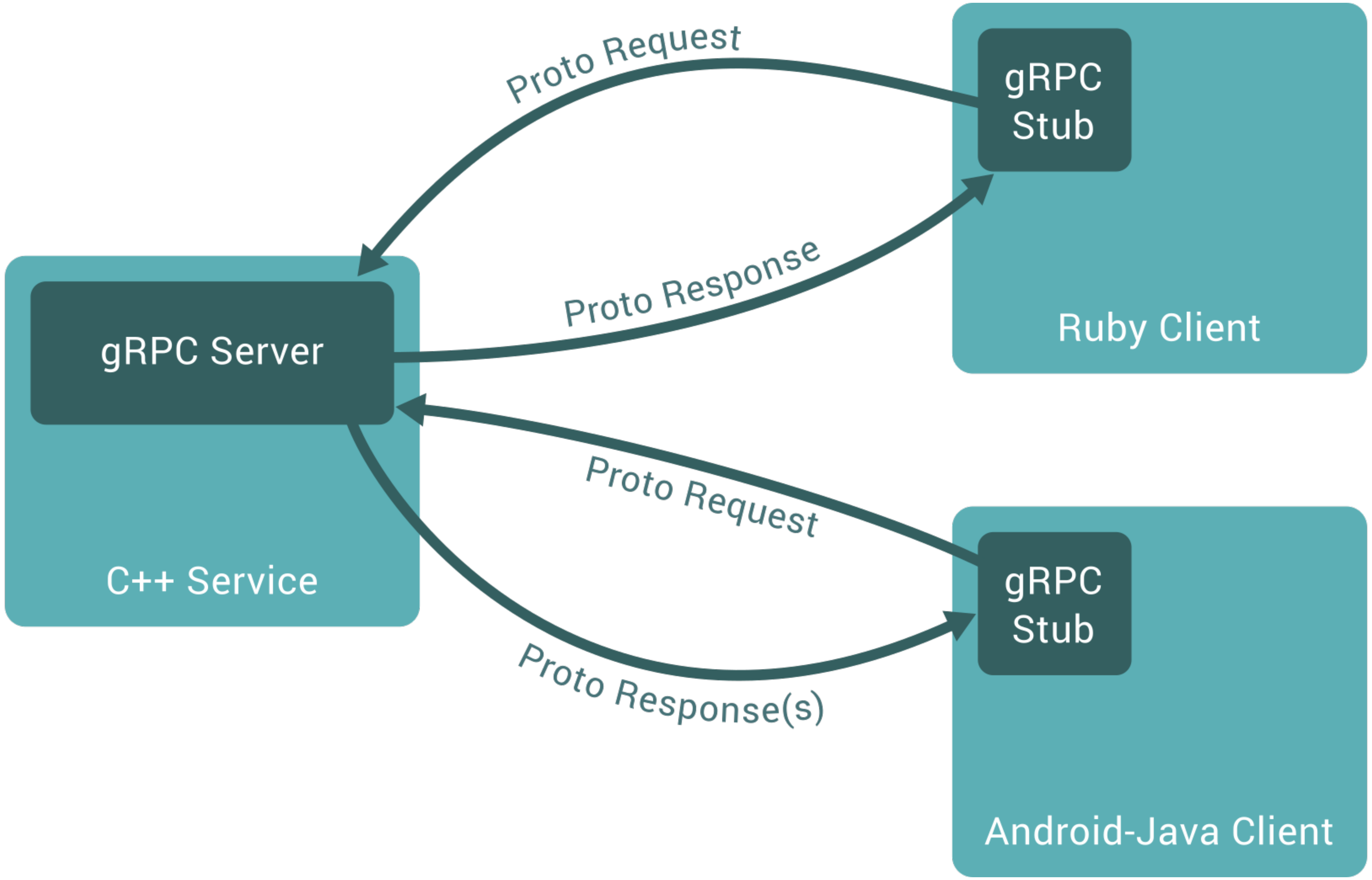
Generate server code

```
$ python -m grpc_tools.protoc \
      --proto_path=../protos \
      --python_out=. \
      --grpc_python_out=. \
      ../protos/route_planner.proto
```

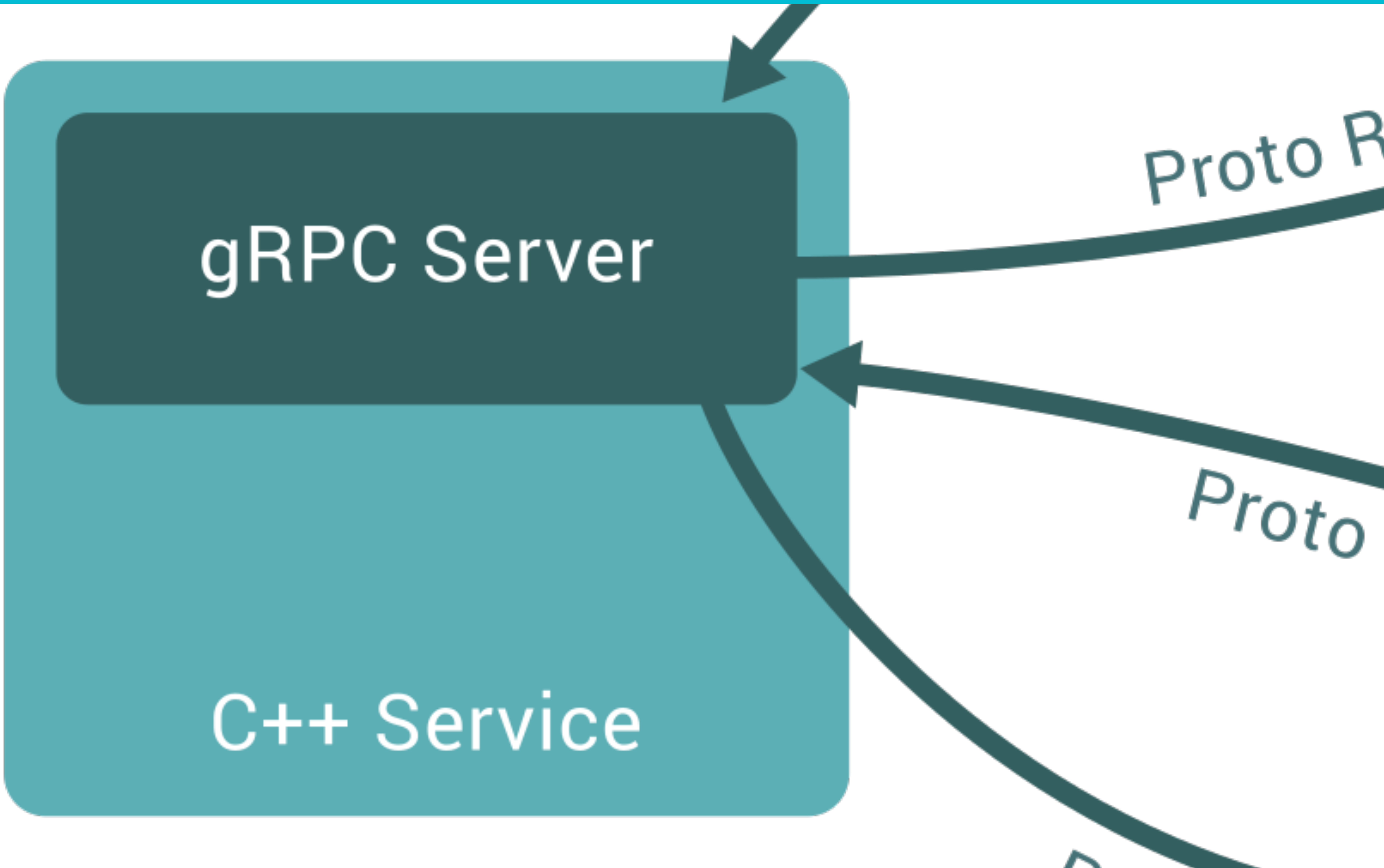
Generated code

```
$ tree
.
├── route_planner_pb2.py
└── route_planner_pb2_grpc.py

0 directories, 2 files
```

Implementing the service



Implementing the service



Service code

```python
class Servicer(route_planner_pb2_grpc.RoutePlannerServicer):
    """Service implementation."""

    def GetRoutes(self, request, context):
        return process_magically_the_request(request)
```

Service code

```
server = grpc.server(
    futures.ThreadPoolExecutor(max_workers=10))

route_planner_pb2_grpc.add_RoutePlannerServicer_to_server(
    Servicer(), server)

server.add_insecure_port('[::]:12345')
server.start()
```

Service code

```
server = grpc.server(
    futures.ThreadPoolExecutor(max_workers=10))

route_planner_pb2_grpc.add_RoutePlannerServicer_to_server(
    Servicer(), server)

server.add_insecure_port('[::]:12345')
server.start()
```

Service code

```
server = grpc.server(
    futures.ThreadPoolExecutor(max_workers=10))

route_planner_pb2_grpc.add_RoutePlannerServicer_to_server(
    Servicer(), server)

server.add_insecure_port('[::]:12345')
server.start()
```

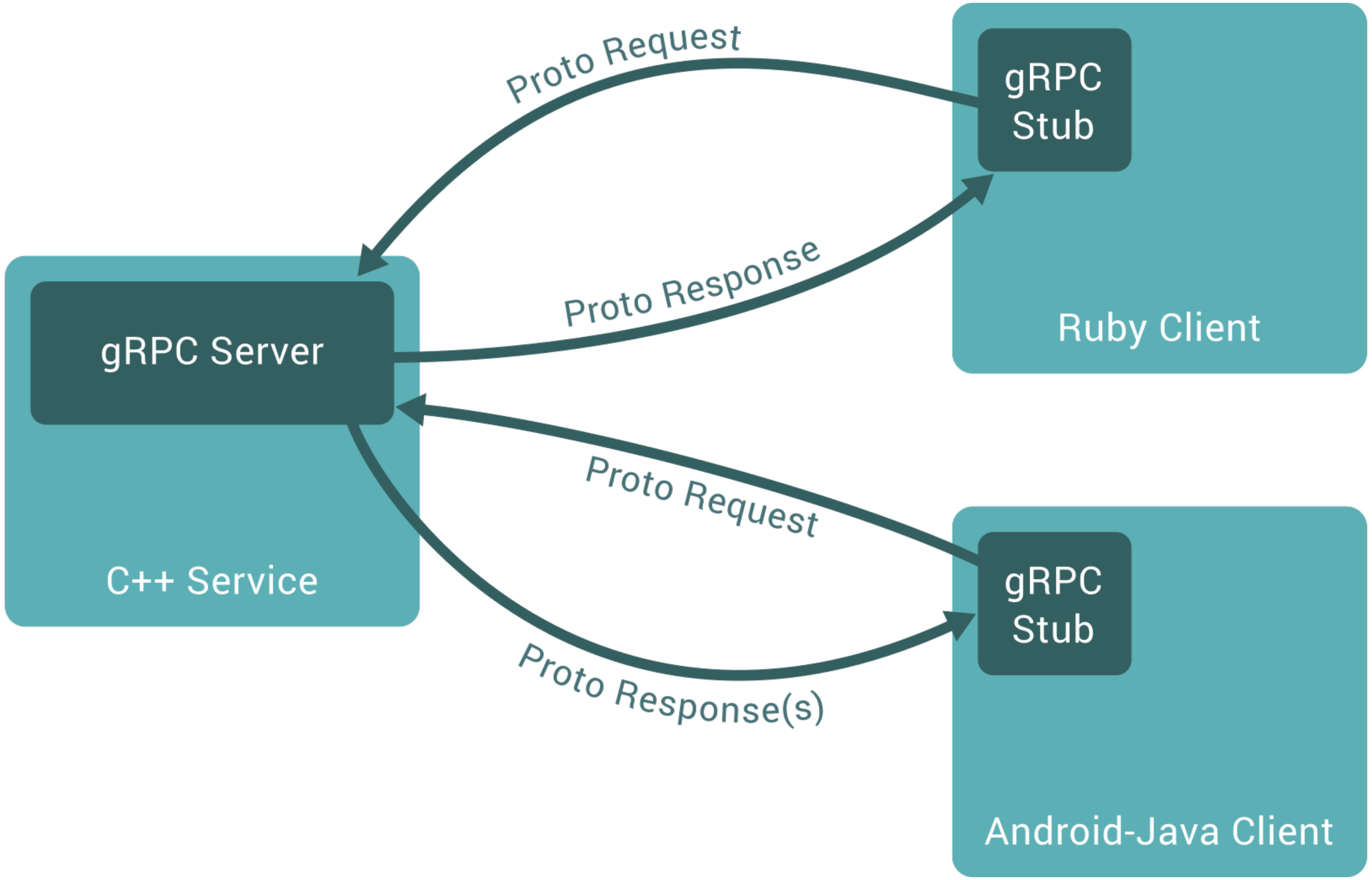
Service code

```
server = grpc.server(
    futures.ThreadPoolExecutor(max_workers=10))

route_planner_pb2_grpc.add_RoutePlannerServicer_to_server(
    Servicer(), server)

server.add_insecure_port('[::]:12345')
server.start()
```

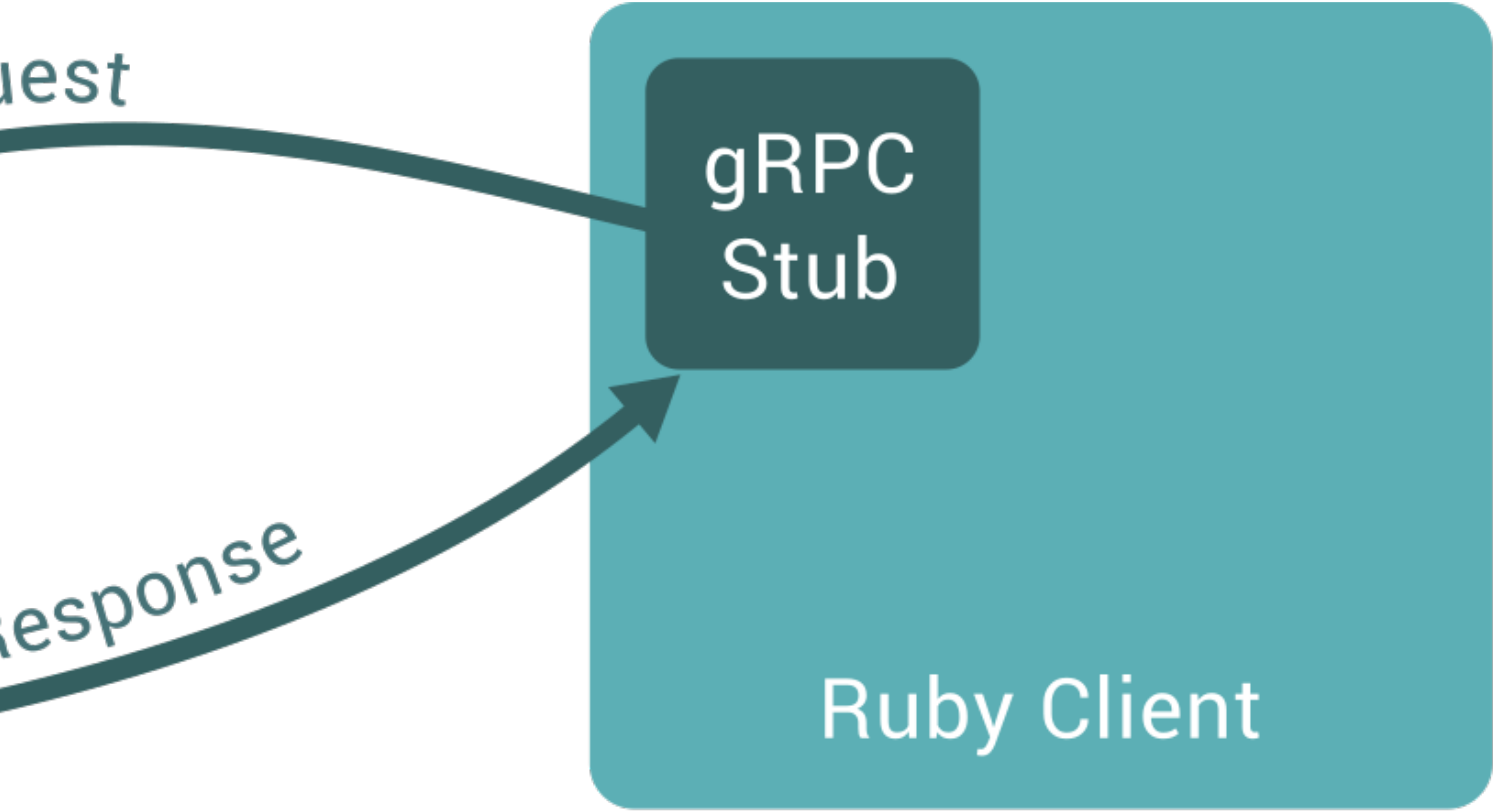
Service code

```
server = grpc.server(
    futures.ThreadPoolExecutor(max_workers=10))

route_planner_pb2_grpc.add_RoutePlannerServicer_to_server(
    Servicer(), server)

server.add_insecure_port('[::]:12345')
server.start()
```

Implementing the client



Implementing the client



Client code

```
channel = grpc.insecure_channel(‘localhost:12345')

stub = route_planner_pb2_grpc.RoutePlannerStub(channel)

request = route_planner_pb2.GetRoutesRequest(
        origin=CURRENT_LOCATION,
        destination=DESTINATION_COORDS)

response = stub.GetRoutes(request)
```

Client code

```
channel = grpc.insecure_channel(‘localhost:12345')

stub = route_planner_pb2_grpc.RoutePlannerStub(channel)

request = route_planner_pb2.GetRoutesRequest(
        origin=CURRENT_LOCATION,
        destination=DESTINATION_COORDS)

response = stub.GetRoutes(request)
```

Client code

```
channel = grpc.insecure_channel(‘localhost:12345')

stub = route_planner_pb2_grpc.RoutePlannerStub(channel)

request = route_planner_pb2.GetRoutesRequest(
        origin=CURRENT_LOCATION,
        destination=DESTINATION_COORDS)

response = stub.GetRoutes(request)
```

Client code

```
channel = grpc.insecure_channel(‘localhost:12345')

stub = route_planner_pb2_grpc.RoutePlannerStub(channel)

request = route_planner_pb2.GetRoutesRequest(
        origin=CURRENT_LOCATION,
        destination=DESTINATION_COORDS)

response = stub.GetRoutes(request)
```

Client code

```
channel = grpc.insecure_channel(‘localhost:12345')

stub = route_planner_pb2_grpc.RoutePlannerStub(channel)

request = route_planner_pb2.GetRoutesRequest(
        origin=CURRENT_LOCATION,
        destination=DESTINATION_COORDS)

response = stub.GetRoutes(request)
```

Client code

```
channel = grpc.insecure_channel(‘localhost:12345')

stub = route_planner_pb2_grpc.RoutePlannerStub(channel)

request = route_planner_pb2.GetRoutesRequest(
        origin=CURRENT_LOCATION,
        destination=DESTINATION_COORDS)

response = stub.GetRoutes(request)
```

Client code (async)

```
response_future = stub.GetRoutes.future(request)

response_future.result()
```

grpc_cli

```
$ grpc_cli call localhost:12345 \
    RoutePlanner.GetRoutes \
    <<- PROTO
        origin: <long: 0.0 lat: 0.0>
        destination: <long: 1.1 lat: 1.1>
    PROTO
```

grpc_cli

```
$ grpc_cli call localhost:12345 \
    RoutePlanner.GetRoutes \
    <<- PROTO
        origin: <long: 0.0 lat: 0.0>
        destination: <long: 1.1 lat: 1.1>
    PROTO
```

grpc_cli

```
$ grpc_cli call localhost:12345 \
    RoutePlanner.GetRoutes \
    <<- PROTO
        origin: <long: 0.0 lat: 0.0>
        destination: <long: 1.1 lat: 1.1>
    PROTO
```

grpc_cli

```
$ grpc_cli call localhost:12345 \
    RoutePlanner.GetRoutes \
    <<- PROTO
        origin: <long: 0.0 lat: 0.0>
        destination: <long: 1.1 lat: 1.1>
    PROTO
```

grpc_cli

```
$ grpc_cli call localhost:12345 \
    RoutePlanner.GetRoutes \
    <<- PROTO
        origin: <long: 0.0 lat: 0.0>
        destination: <long: 1.1 lat: 1.1>
    PROTO
```

grpc_cli

```
Rpc succeeded with OK status
Response:
  routes: <...>
  routes: <...>
  routes: <...>
  routes: <...>
```

Service definition

```
service RoutePlanner {
    rpc GetRoutes (GetRoutesRequest)
        returns (GetRoutesResponse) {}
}
```

Service definition - response streaming

```
service RoutePlanner {
    rpc GetRoutes (GetRoutesRequest)
        returns (stream GetRoutesResponse) {}
}
```

Service definition

```
service RoutePlanner {
    rpc GetRoutes (GetRoutesRequest)
        returns (GetRoutesResponse) {}
}
```

Service definition - request streaming

```
service RoutePlanner {
    rpc GetRoutes (stream GetRoutesRequest)
        returns (GetRoutesResponse) {}
}
```

Request streaming? Response streaming?

Request streaming? Response streaming?

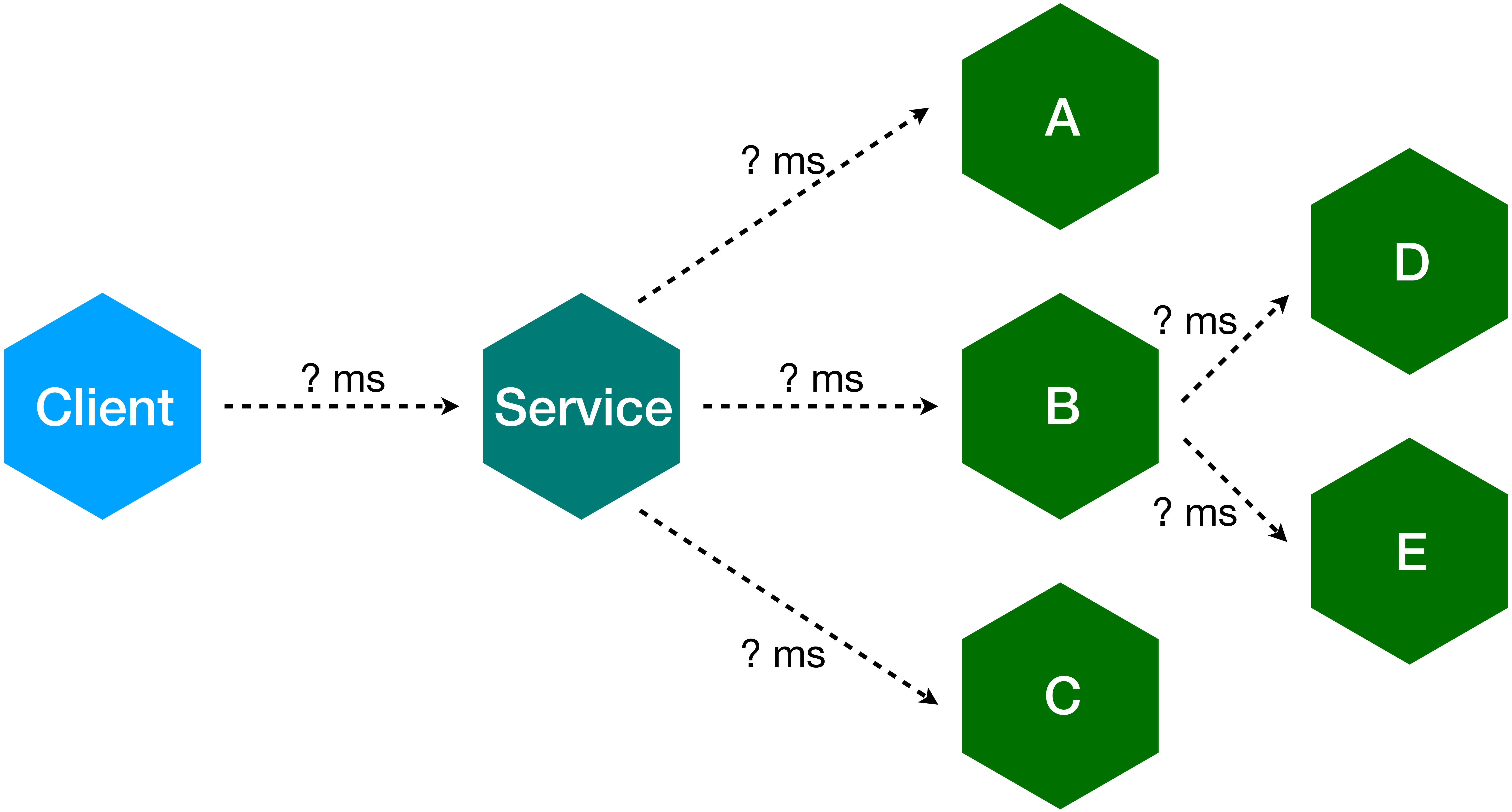
Service definition - bidirectional streaming

```
service RoutePlanner {
    rpc GetRoutes (stream GetRoutesRequest)
        returns (stream GetRoutesResponse) {}
}
```

Keep in mind…

Things will go wrong

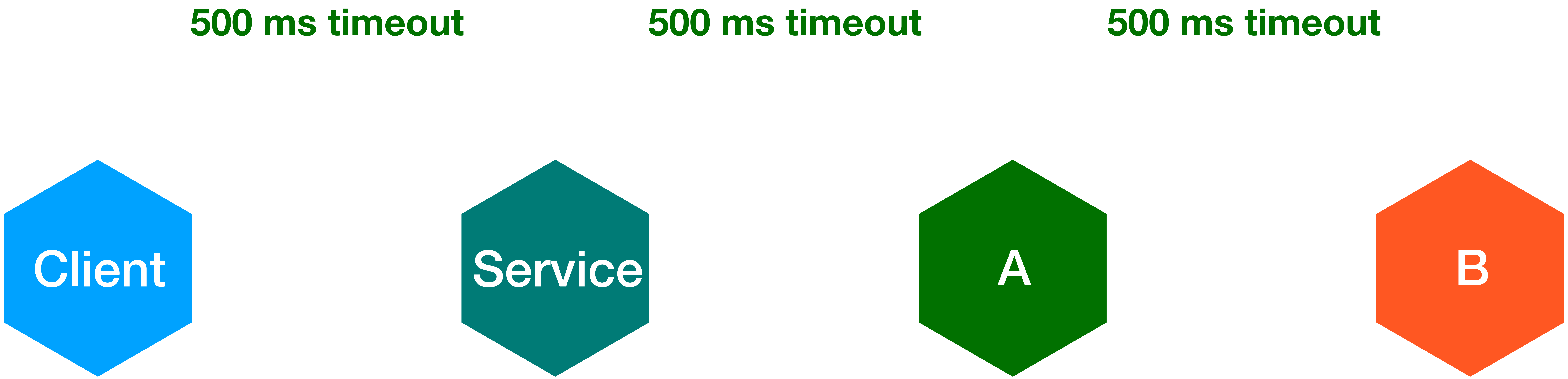
Timeouts



Uniform timeout

Client
Service
A
B

Uniform timeout



500 ms timeout

500 ms timeout

500 ms timeout

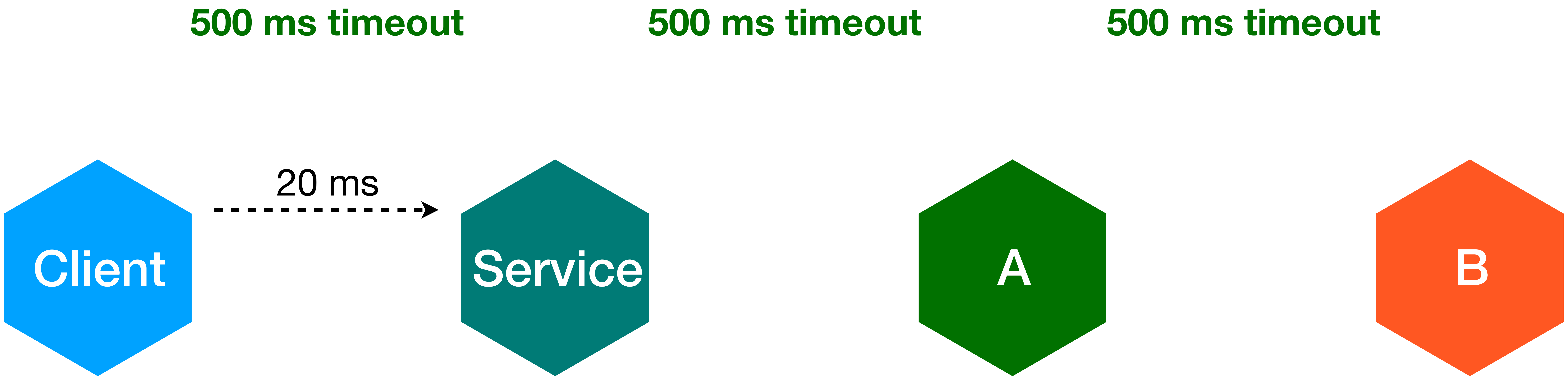
Client

Service

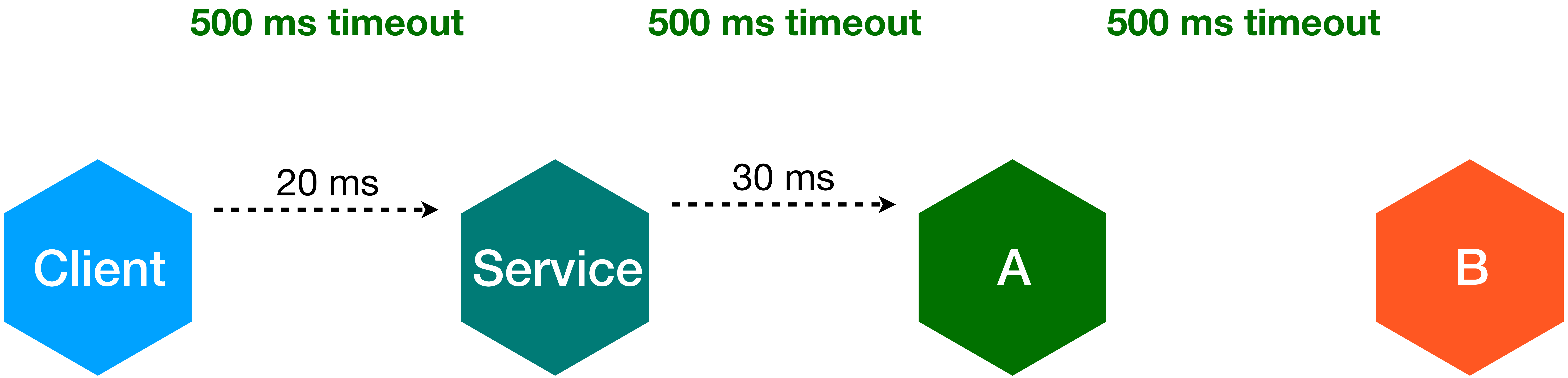
A

B

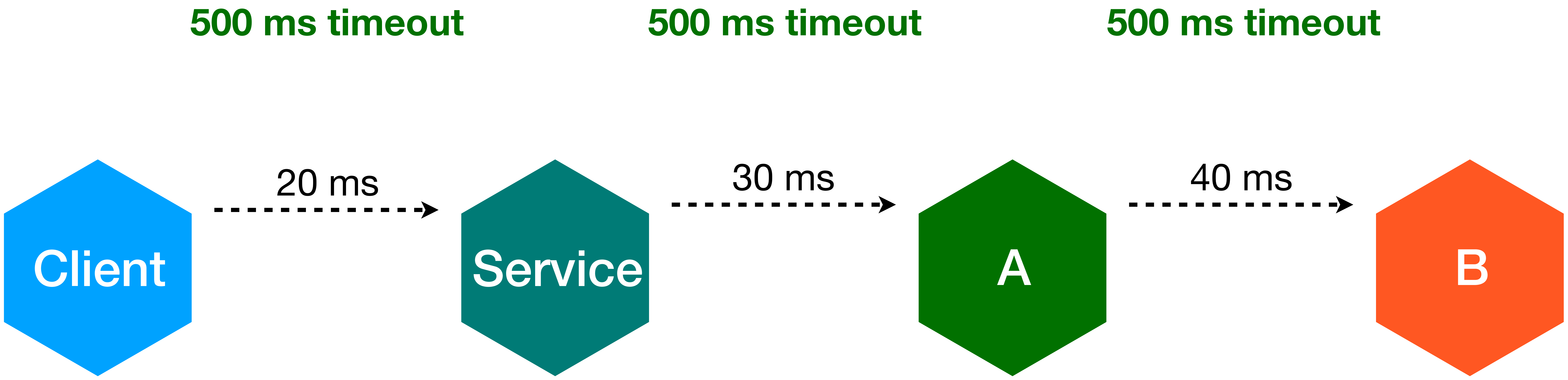
Uniform timeout



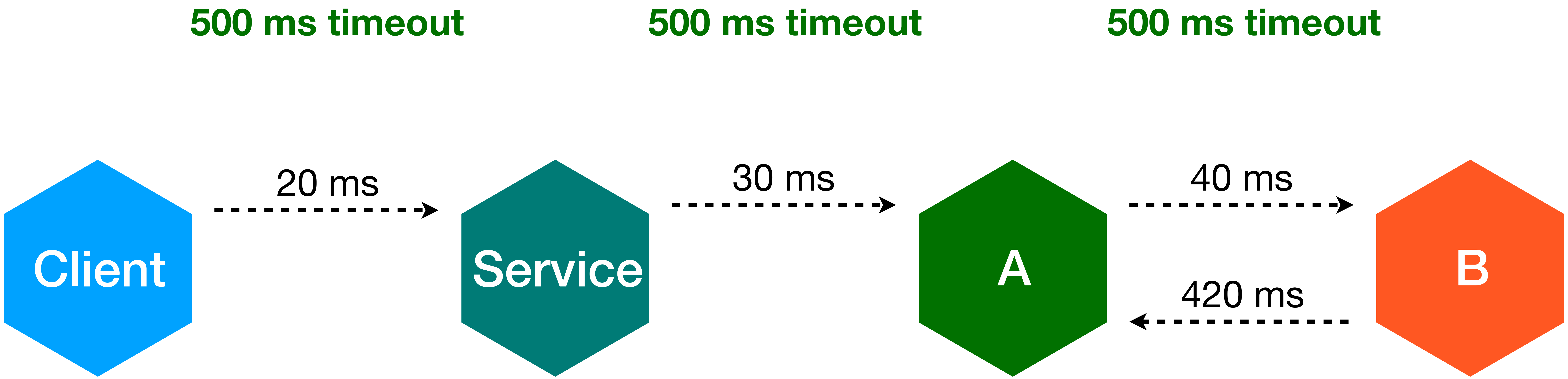
Uniform timeout



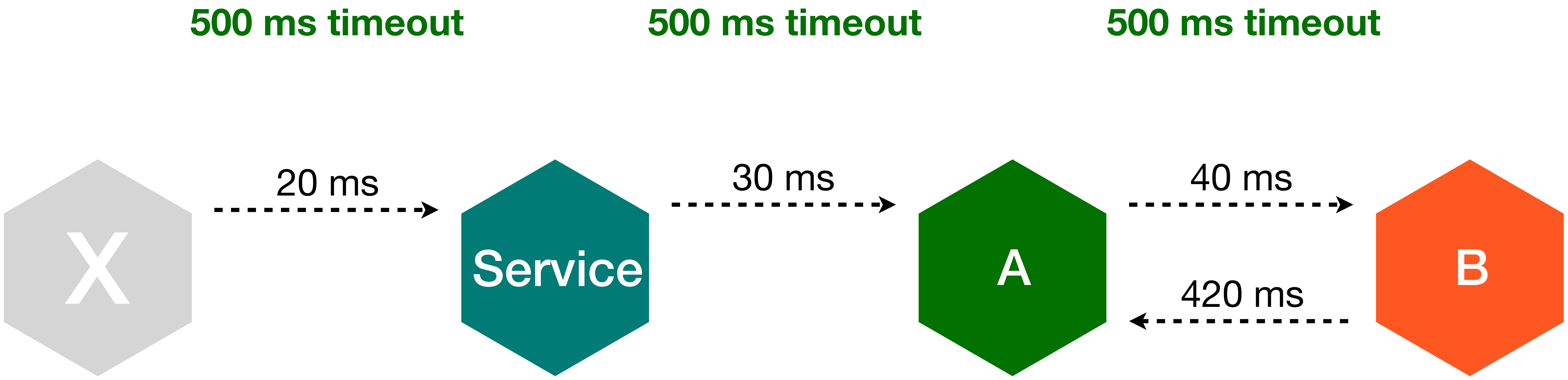
Uniform timeout



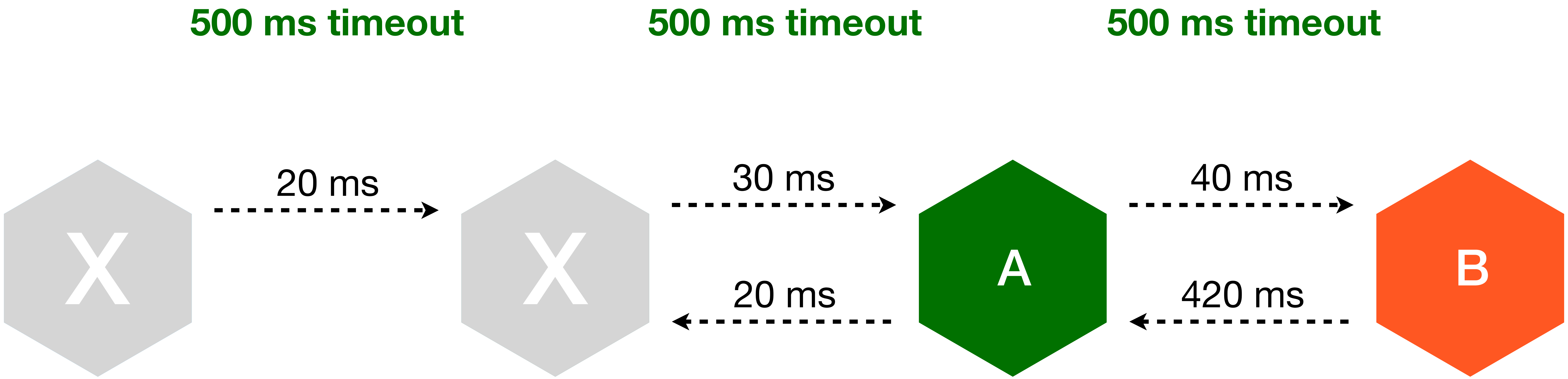
Uniform timeout



Uniform timeout



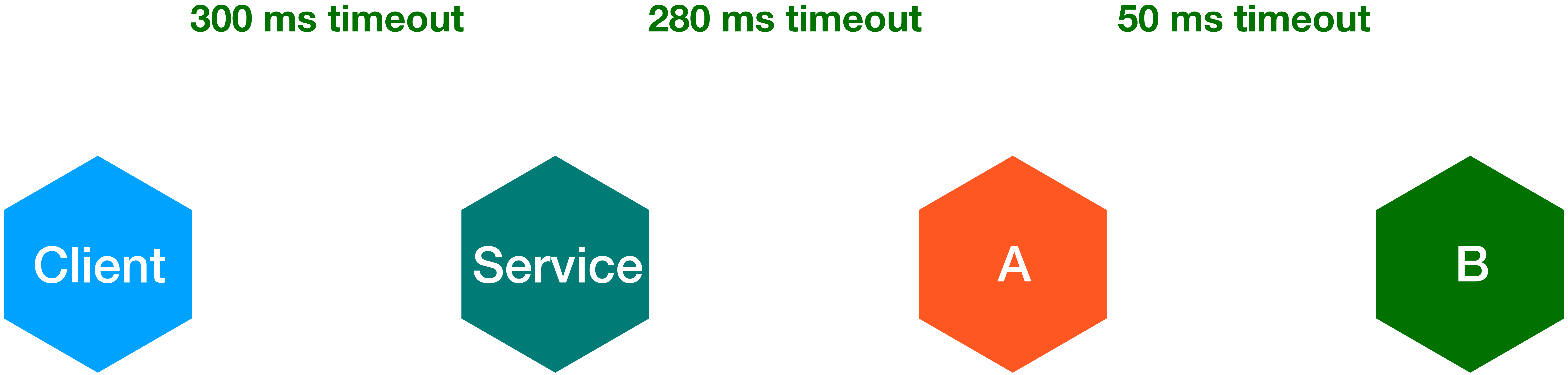
Uniform timeout



Fine-tuned timeout

Client
Service
A
B

Fine-tuned timeout



300 ms timeout

280 ms timeout

50 ms timeout

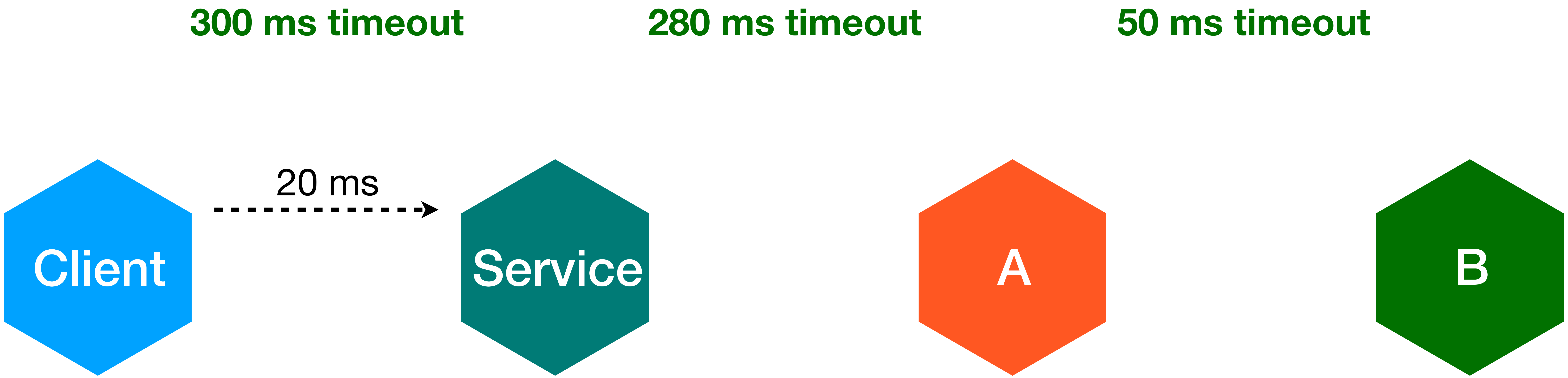
Client

Service

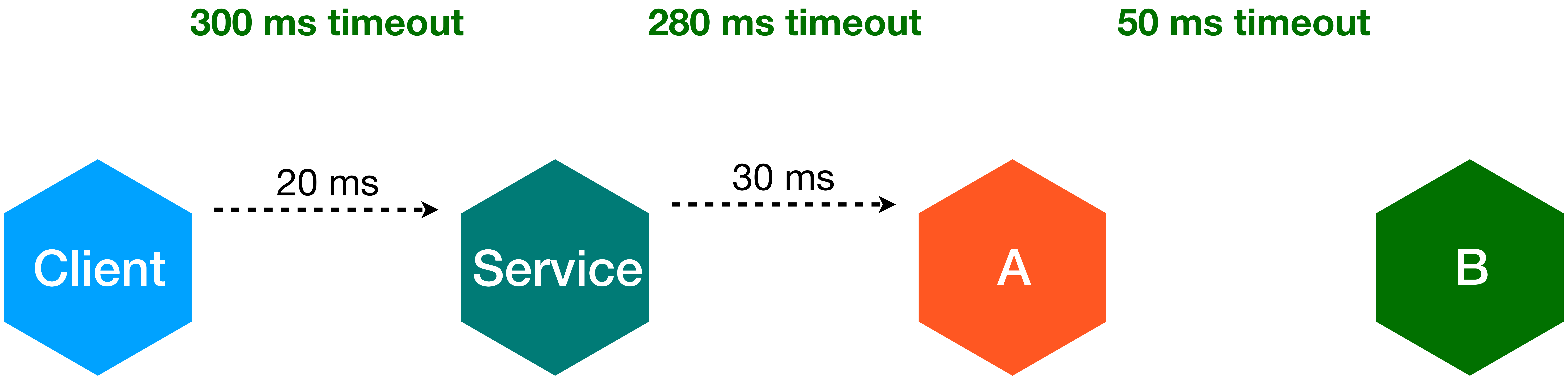
A

B

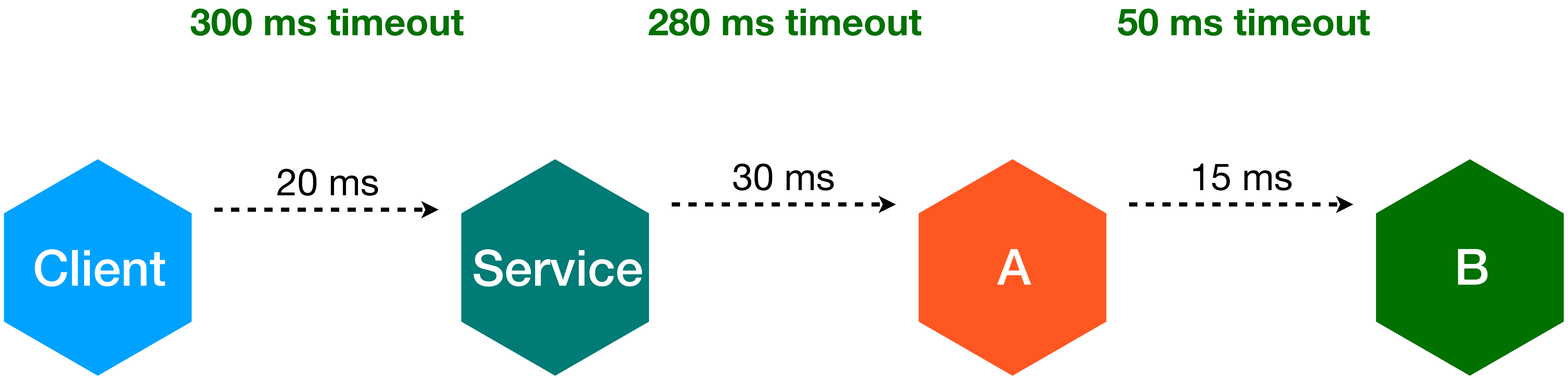
Fine-tuned timeout



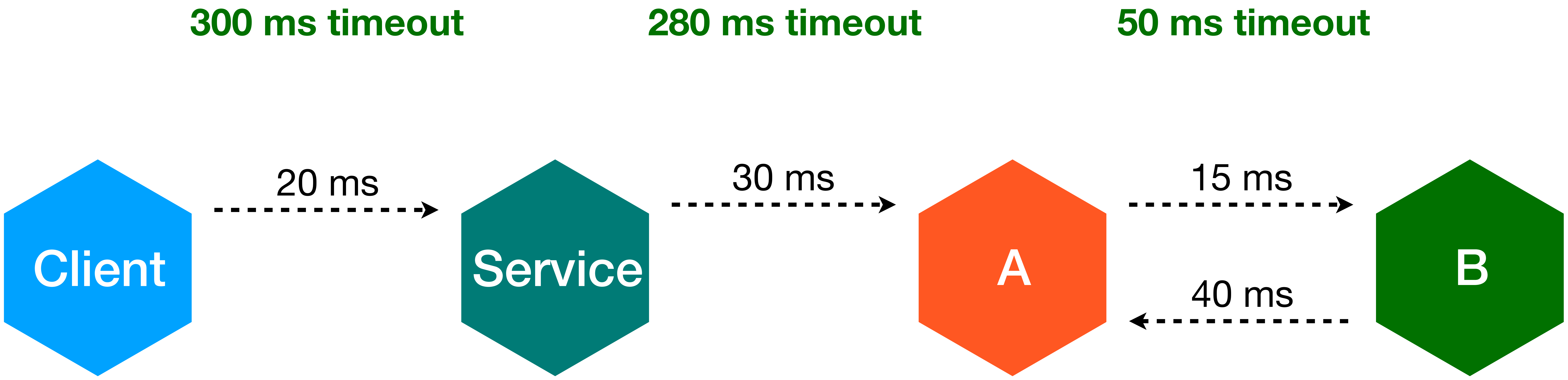
Fine-tuned timeout



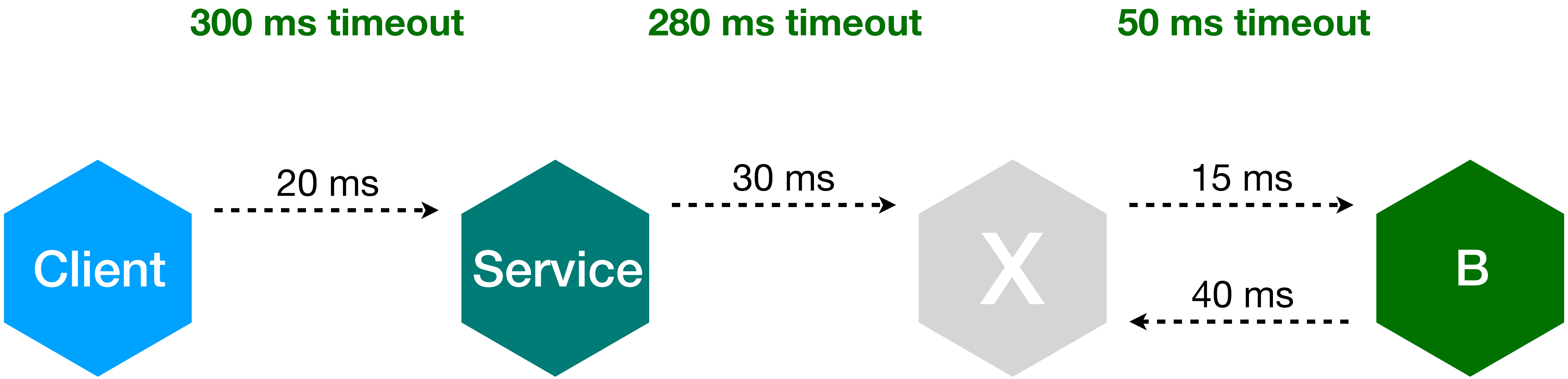
Fine-tuned timeout



Fine-tuned timeout



Fine-tuned timeout



Adaptive timeout

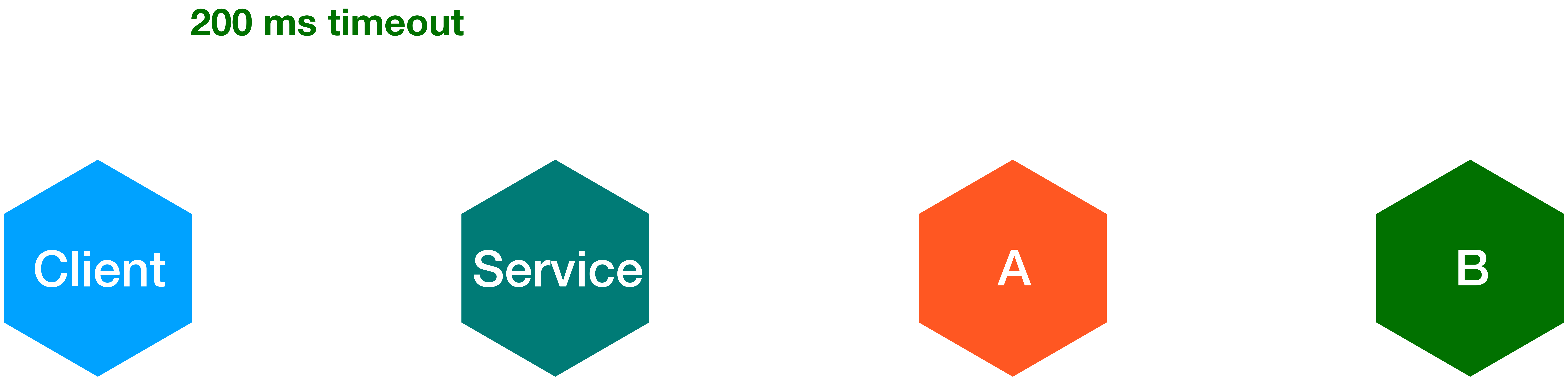
Client

Service

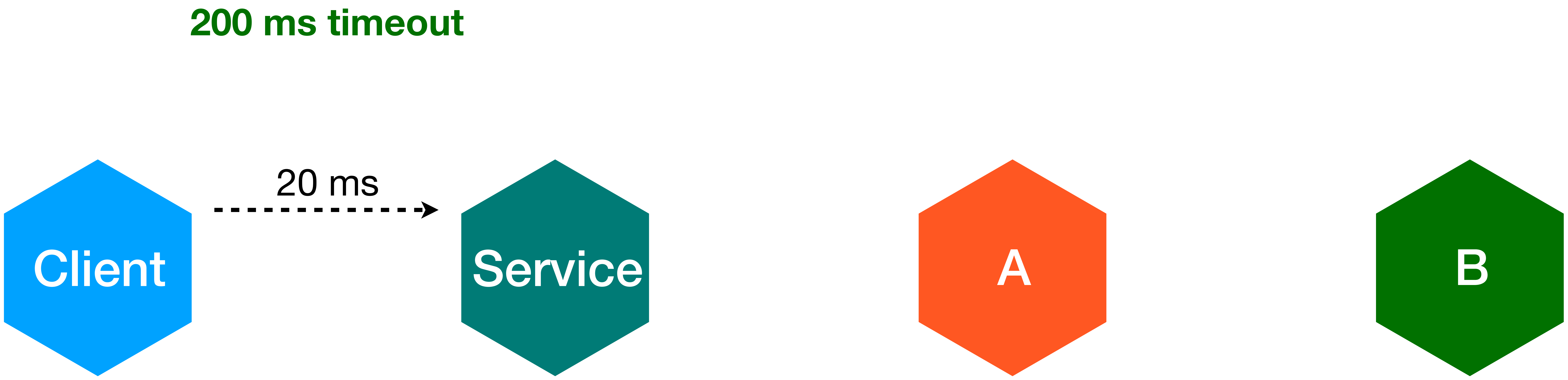
A

B

Adaptive timeout



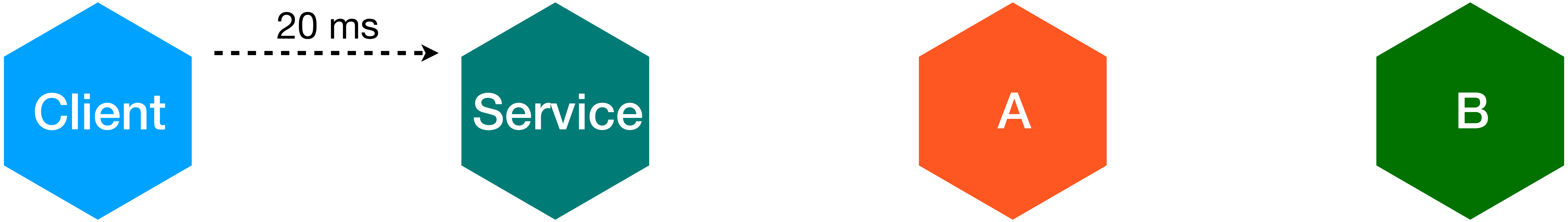
Adaptive timeout



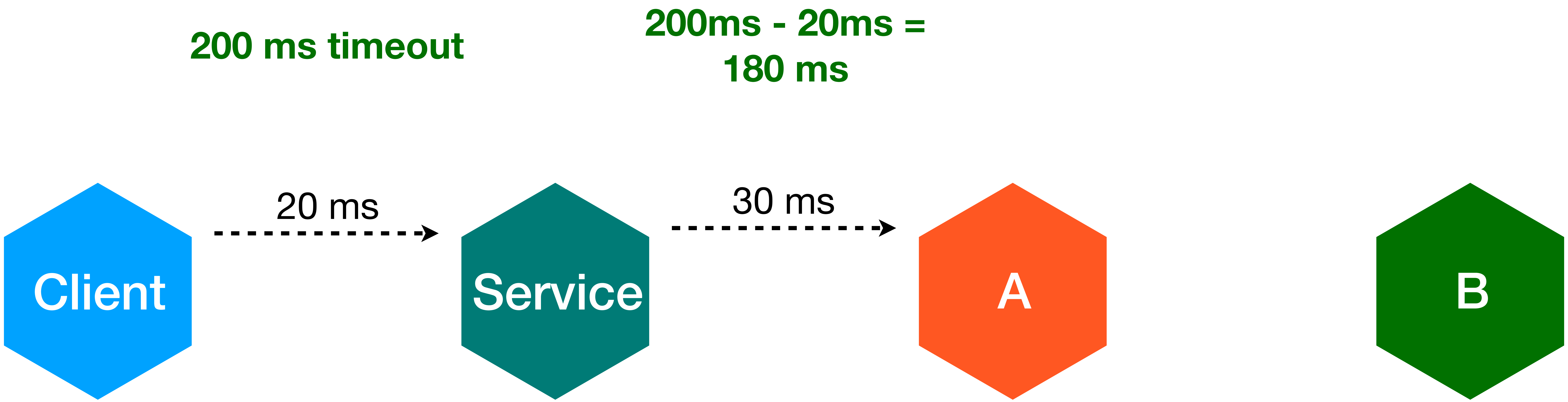
Adaptive timeout

200 ms timeout

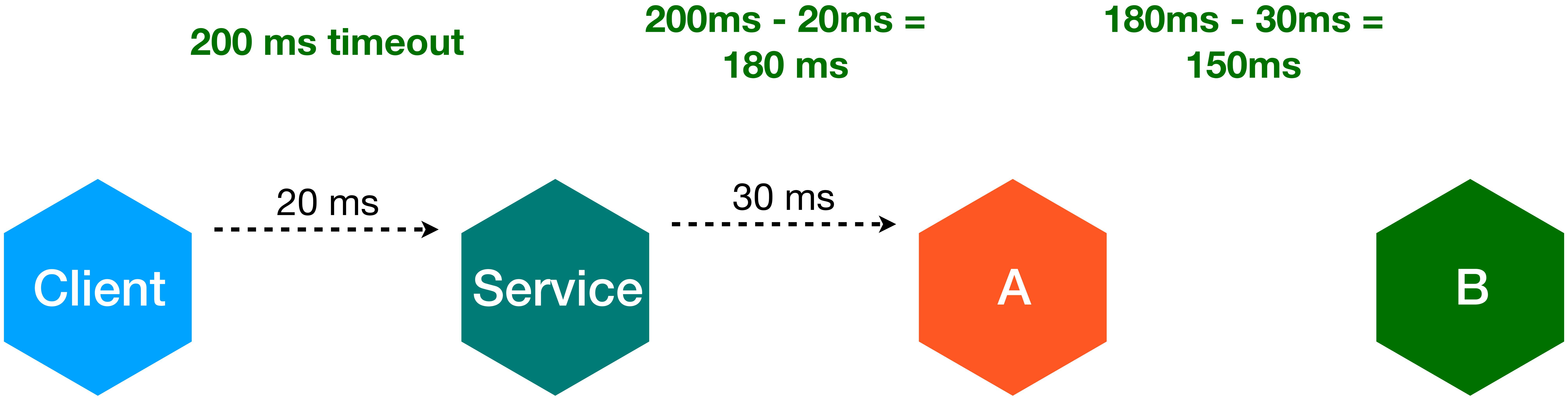
200ms - 20ms =
180 ms



Adaptive timeout



Adaptive timeout

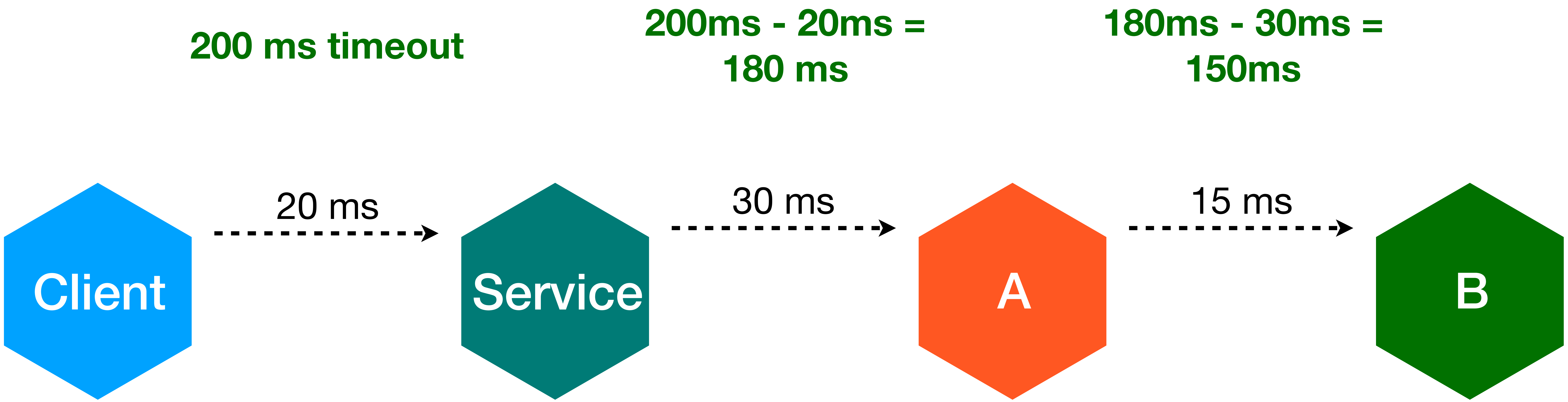


200 ms timeout

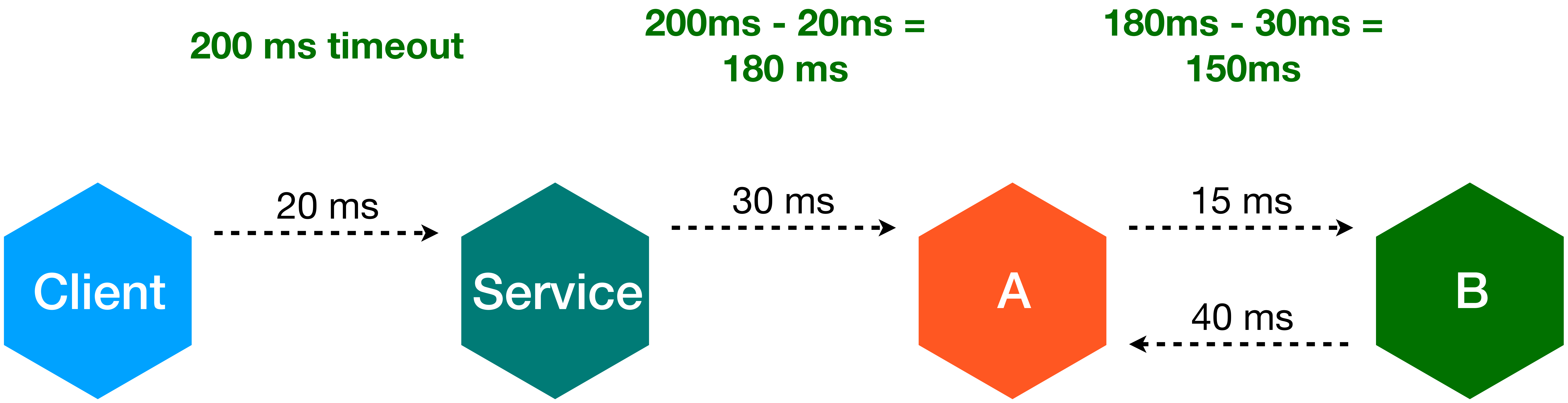
200ms - 20ms =
180 ms

180ms - 30ms =
150ms

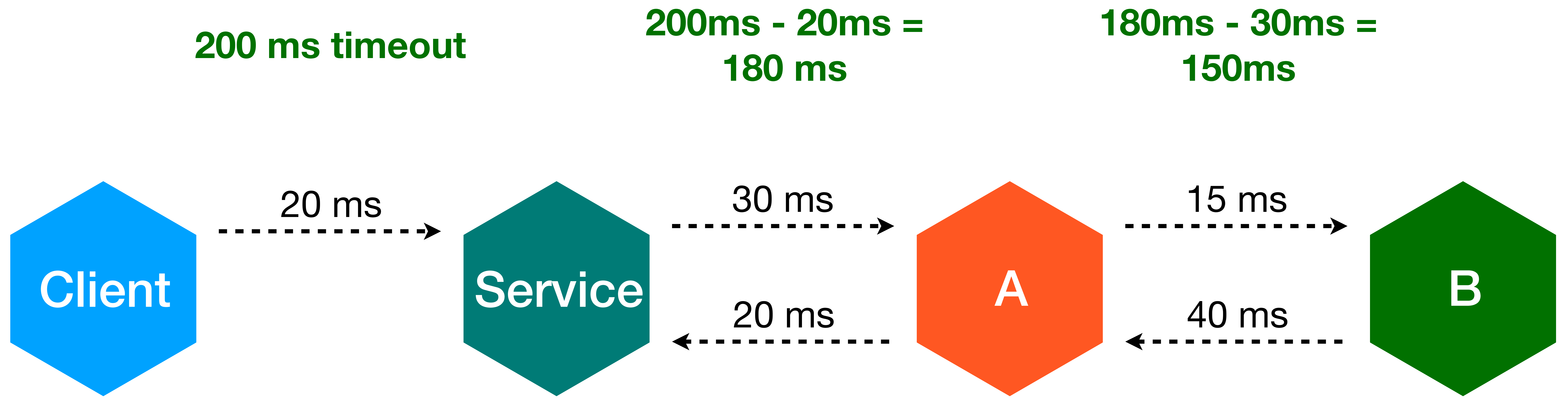
Adaptive timeout



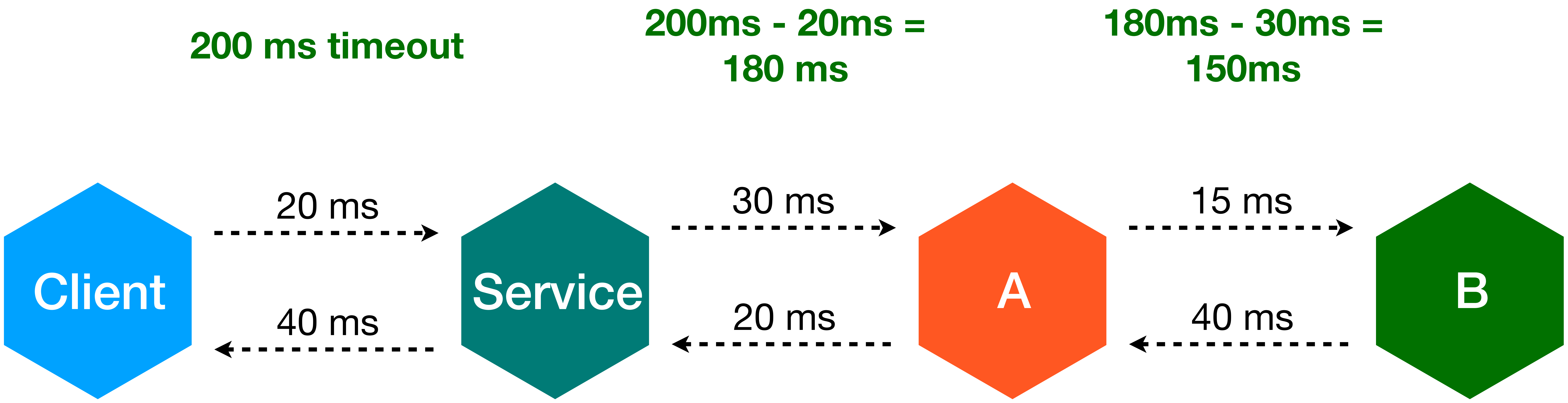
Adaptive timeout



Adaptive timeout



Adaptive timeout



gRPC: Deadlines

gRPC: Deadlines

– Timeout is relative

gRPC: Deadlines

– Timeout is relative

– Deadline is absolute

Deadline propagation

```
Start TS: 3600000
Timeout:      200
Deadline: 3600200
```

Client
Service
A
B

Deadline propagation

Start TS: 3600**000**
Timeout: 200
Deadline: 3600**200**

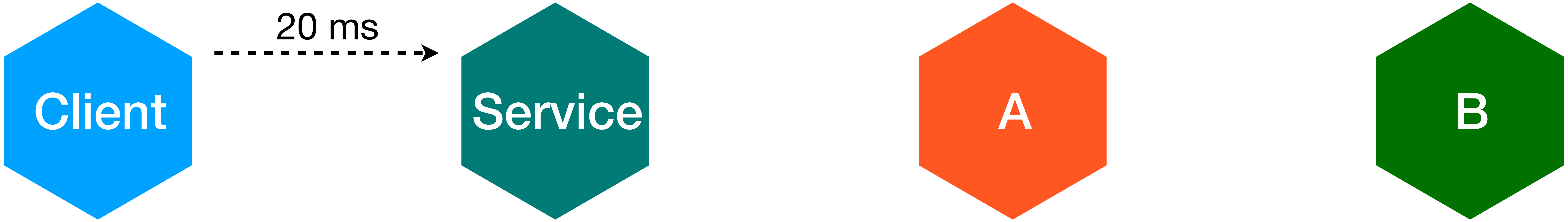
TS: 3600**000**

Client
Service
A
B

Deadline propagation

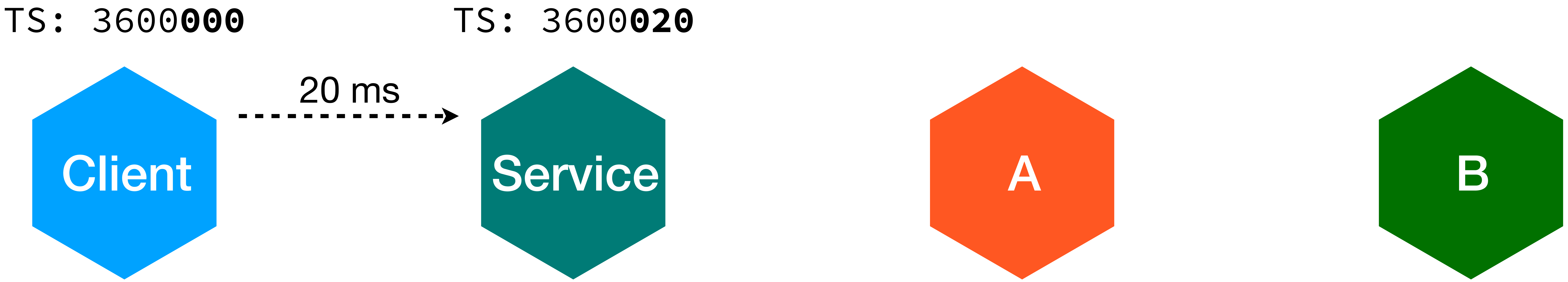
```
Start TS: 3600**000**
Timeout:       200
Deadline: 3600**200**
```

TS: 3600**000**



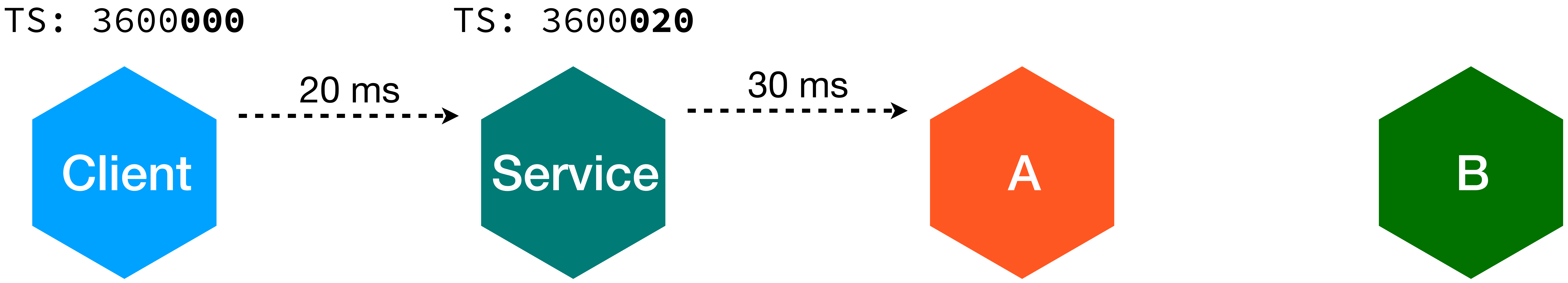
Deadline propagation

```
Start TS: 3600000
Timeout:      200
Deadline: 3600200
```



Deadline propagation

Start TS: 3600**000**
Timeout: 200
Deadline: 3600**200**



TS: 3600**000**

TS: 3600**020**

20 ms

30 ms

Client

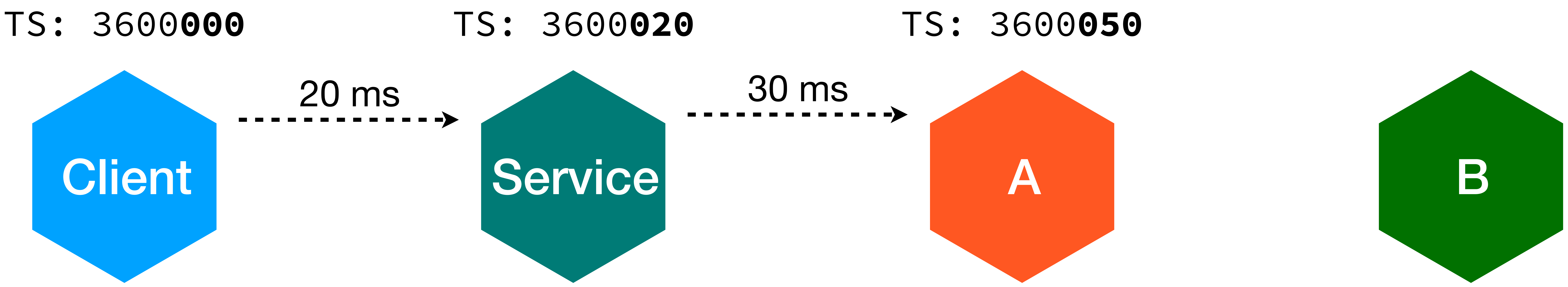
Service

A

B

Deadline propagation

Start TS: 3600**000**
Timeout: 200
Deadline: 3600**200**



TS: 3600**000**

TS: 3600**020**

TS: 3600**050**

20 ms

30 ms

Client

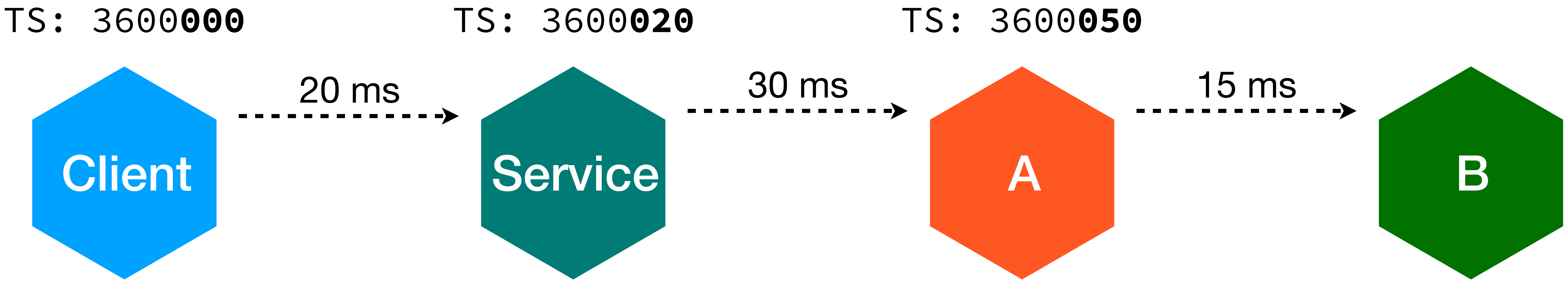
Service

A

B

Deadline propagation

```
Start TS: 3600000
Timeout:      200
Deadline: 3600200
```



TS: 3600000

Client

20 ms

TS: 3600020

Service

30 ms

TS: 3600050

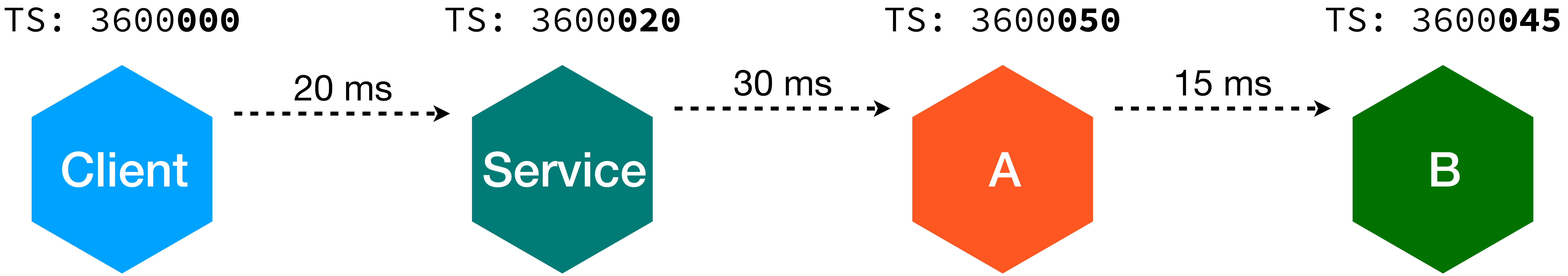
A

15 ms

B

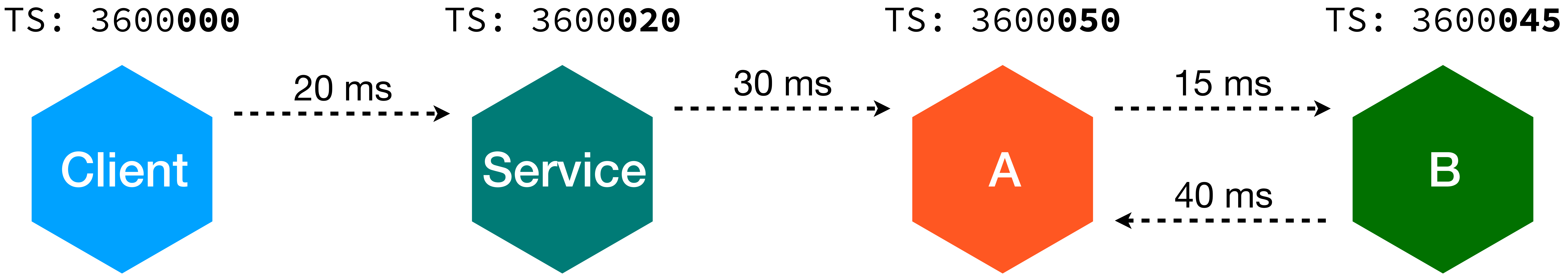
Deadline propagation

```
Start TS: 3600000
Timeout:      200
Deadline: 3600200
```



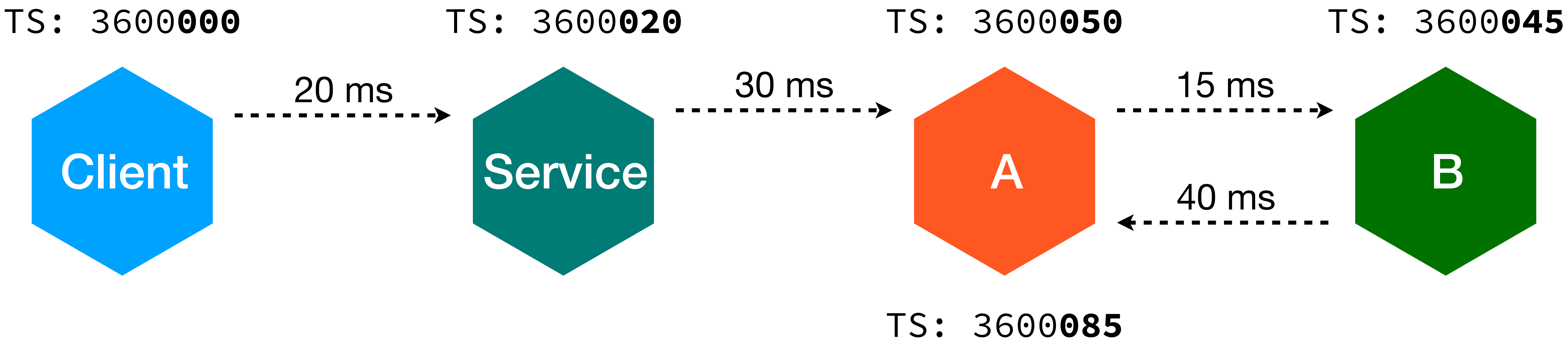
Deadline propagation

```
Start TS: 3600000
Timeout:      200
Deadline: 3600200
```



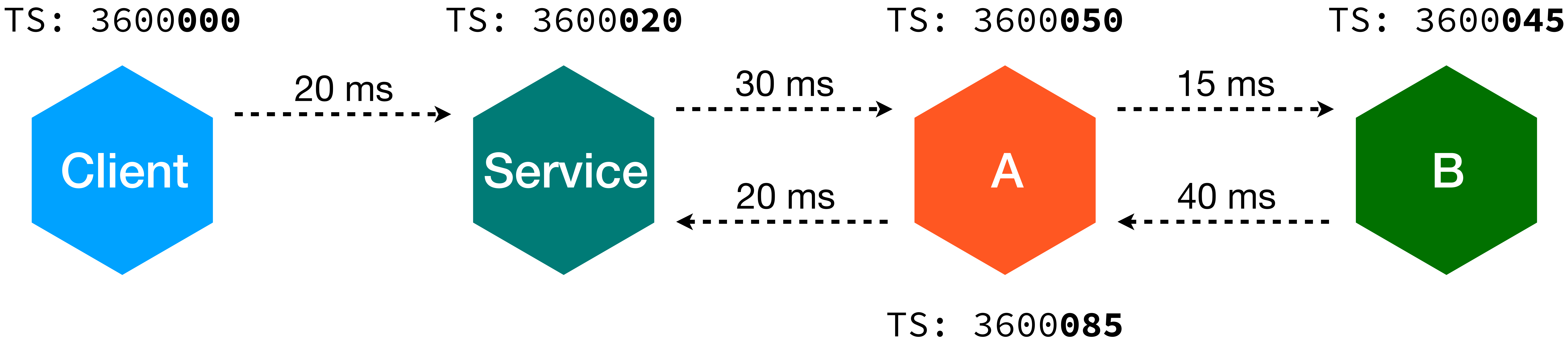
Deadline propagation

```
Start TS: 3600000
Timeout:      200
Deadline: 3600200
```



Deadline propagation

```
Start TS: 3600000
Timeout:      200
Deadline: 3600200
```



TS: 3600000

Client

20 ms

TS: 3600020

Service

30 ms

20 ms

TS: 3600050

A

TS: 3600085

15 ms

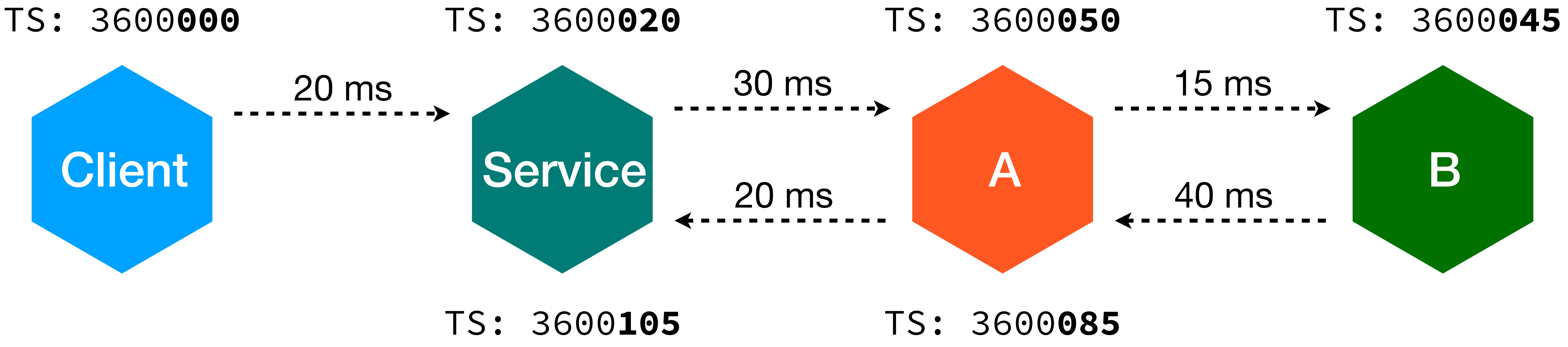
40 ms

TS: 3600045

B

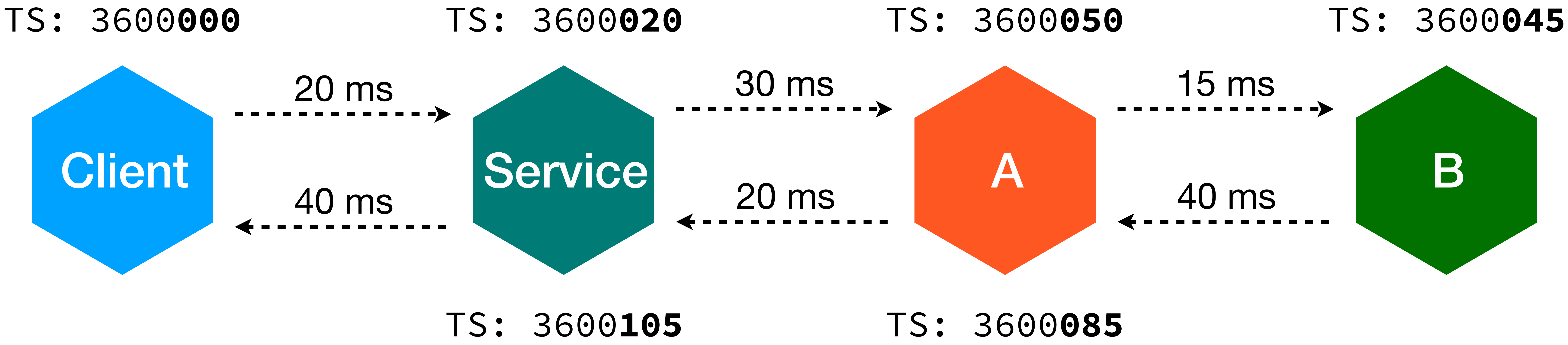
Deadline propagation

```
Start TS: 3600000
Timeout:      200
Deadline: 3600200
```



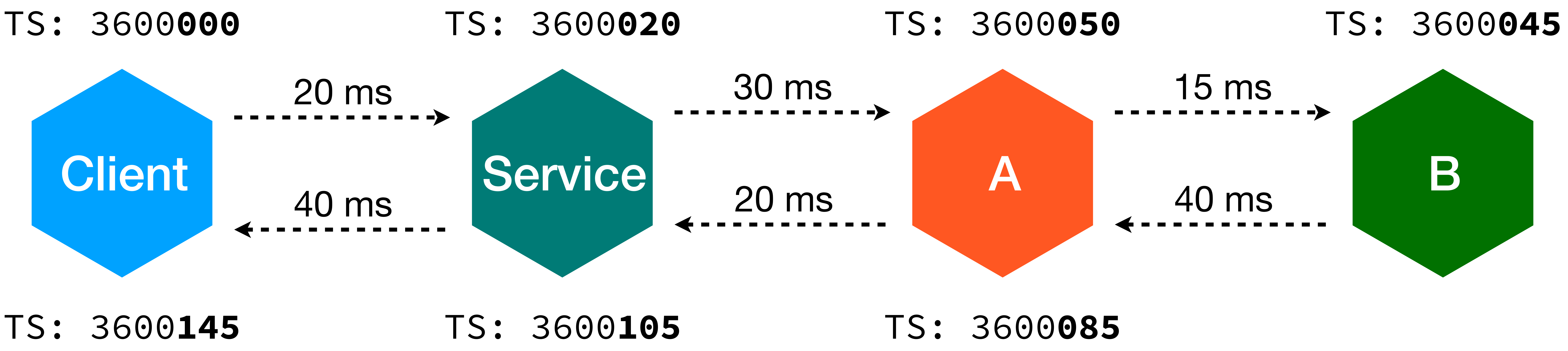
Deadline propagation

```
Start TS: 3600000
Timeout:      200
Deadline: 3600200
```



Deadline propagation

```
Start TS: 3600000
Timeout:      200
Deadline: 3600200
```



TS: 3600000 TS: 3600020 TS: 3600050 TS: 3600045

Client → 20 ms → Service → 30 ms → A → 15 ms → B

B → 40 ms → A → 20 ms → Service → 40 ms → Client

TS: 3600145 TS: 3600105 TS: 3600085

Deadline propagation

Start TS: 3600**000**
Timeout: 200
Deadline: 3600**200**

Client
Service
A
B

Deadline propagation

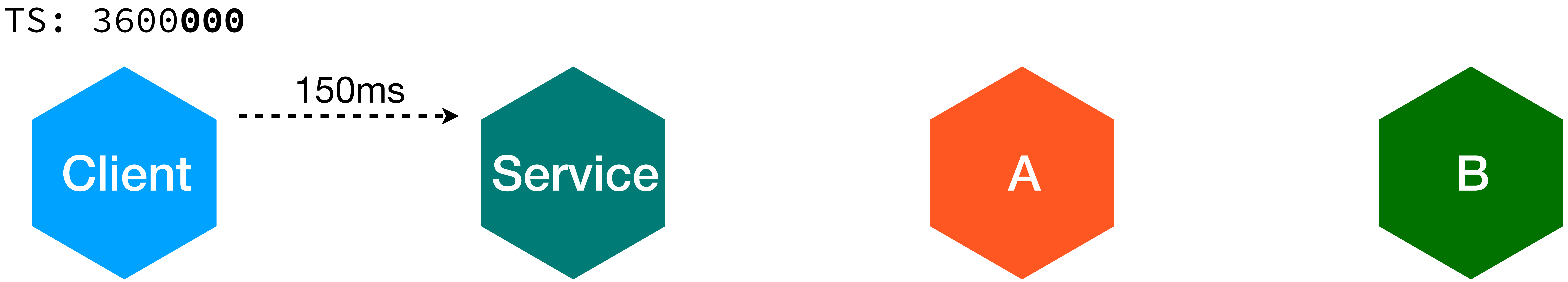
Start TS: 3600**000**
Timeout: 200
Deadline: 3600**200**

TS: 3600**000**

Client
Service
A
B

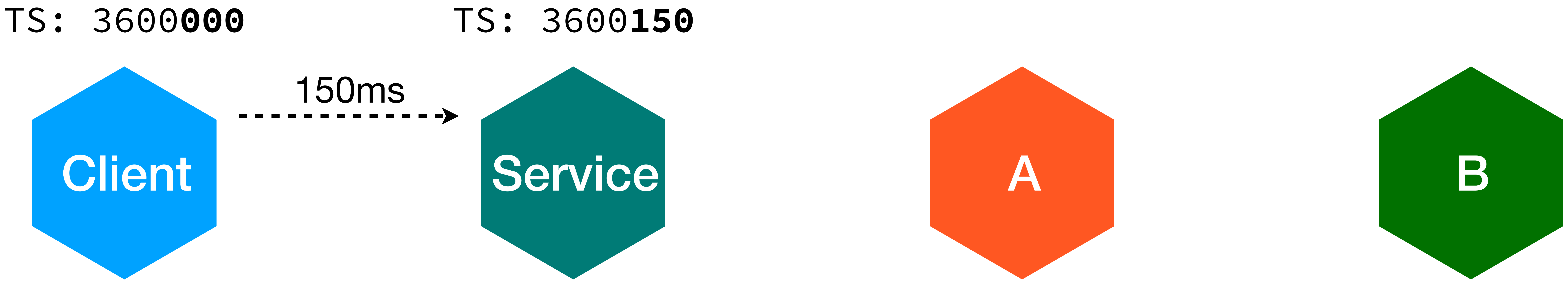
Deadline propagation

```
Start TS: 3600000
Timeout:      200
Deadline: 3600200
```



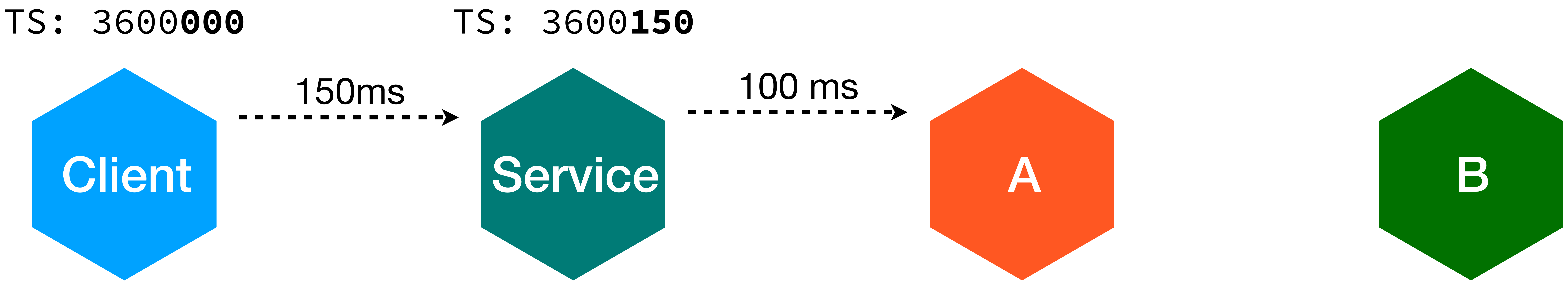
Deadline propagation

```
Start TS: 3600000
Timeout:      200
Deadline: 3600200
```



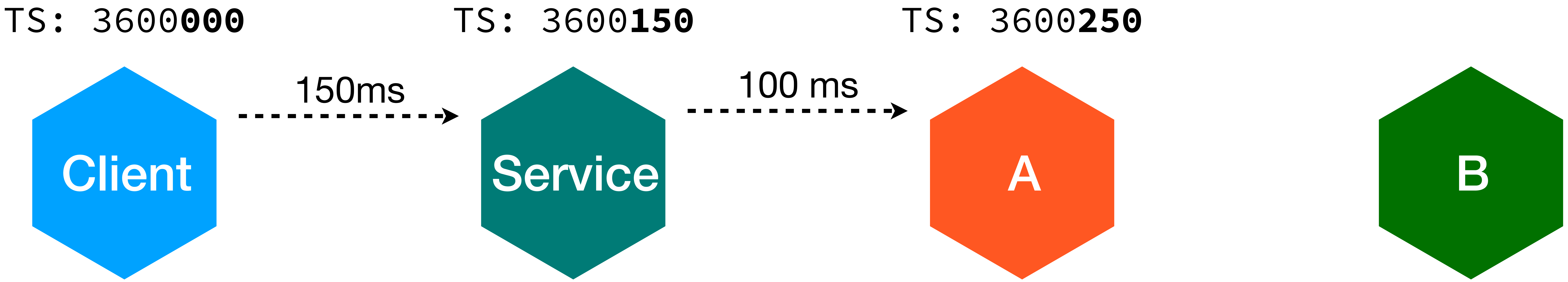
Deadline propagation

```
Start TS: 3600000
Timeout:      200
Deadline: 3600200
```



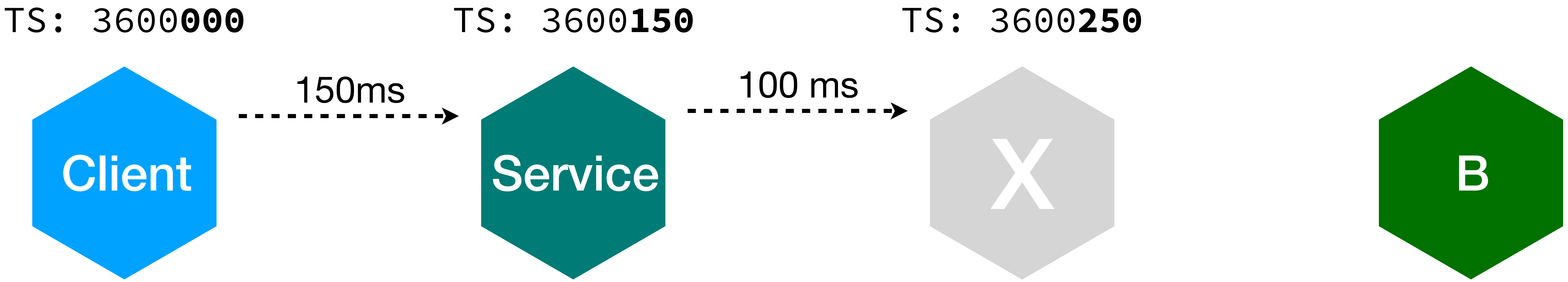
Deadline propagation

```
Start TS: 3600000
Timeout:      200
Deadline: 3600200
```



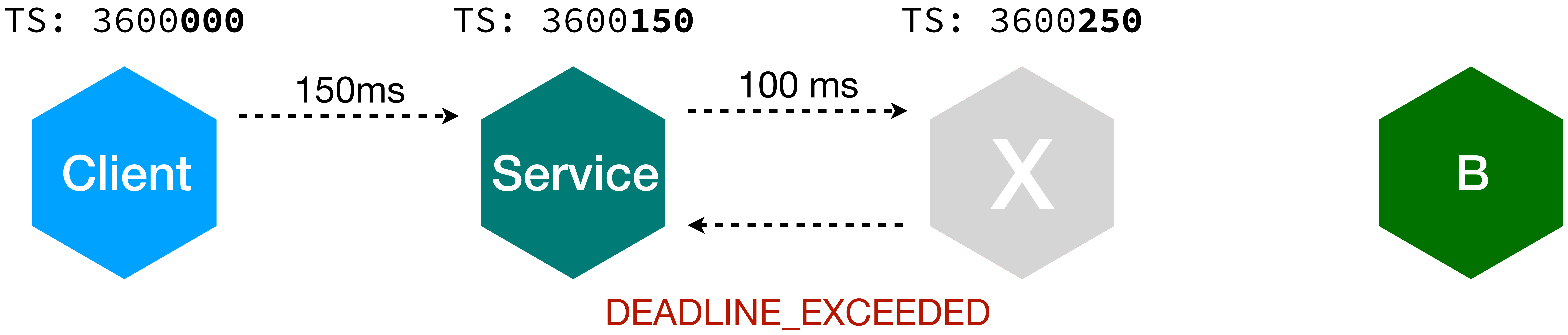
Deadline propagation

```
Start TS: 3600000
Timeout:      200
Deadline: 3600200
```



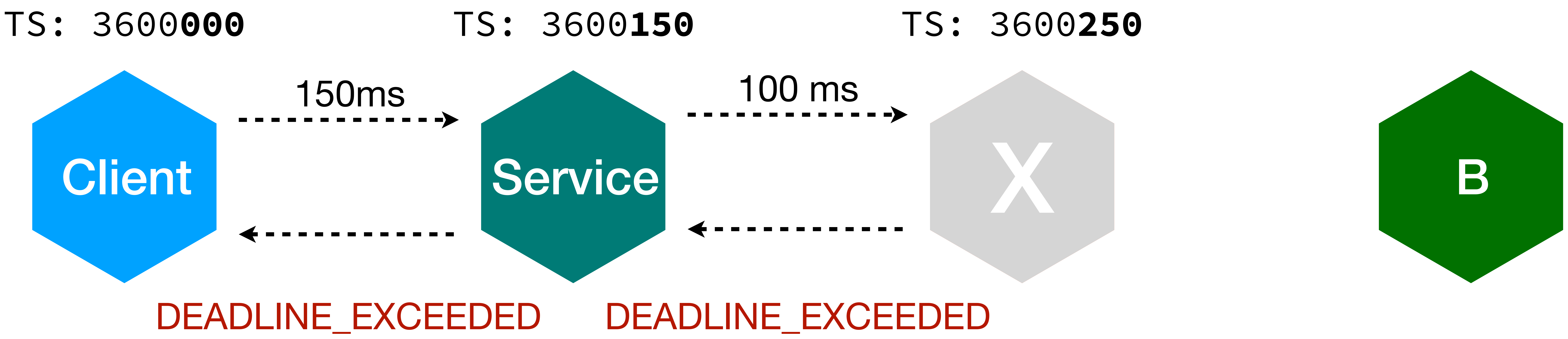
Deadline propagation

```
Start TS: 3600000
Timeout:      200
Deadline: 3600200
```



Deadline propagation

```
Start TS: 3600000
Timeout:      200
Deadline: 3600200
```



Cancellation

Cancellation

- Can be initiated by both client/server

Cancellation

- Can be initiated by both client/server

- Immediately terminates the RPC

Cancellation

- Can be initiated by both client/server

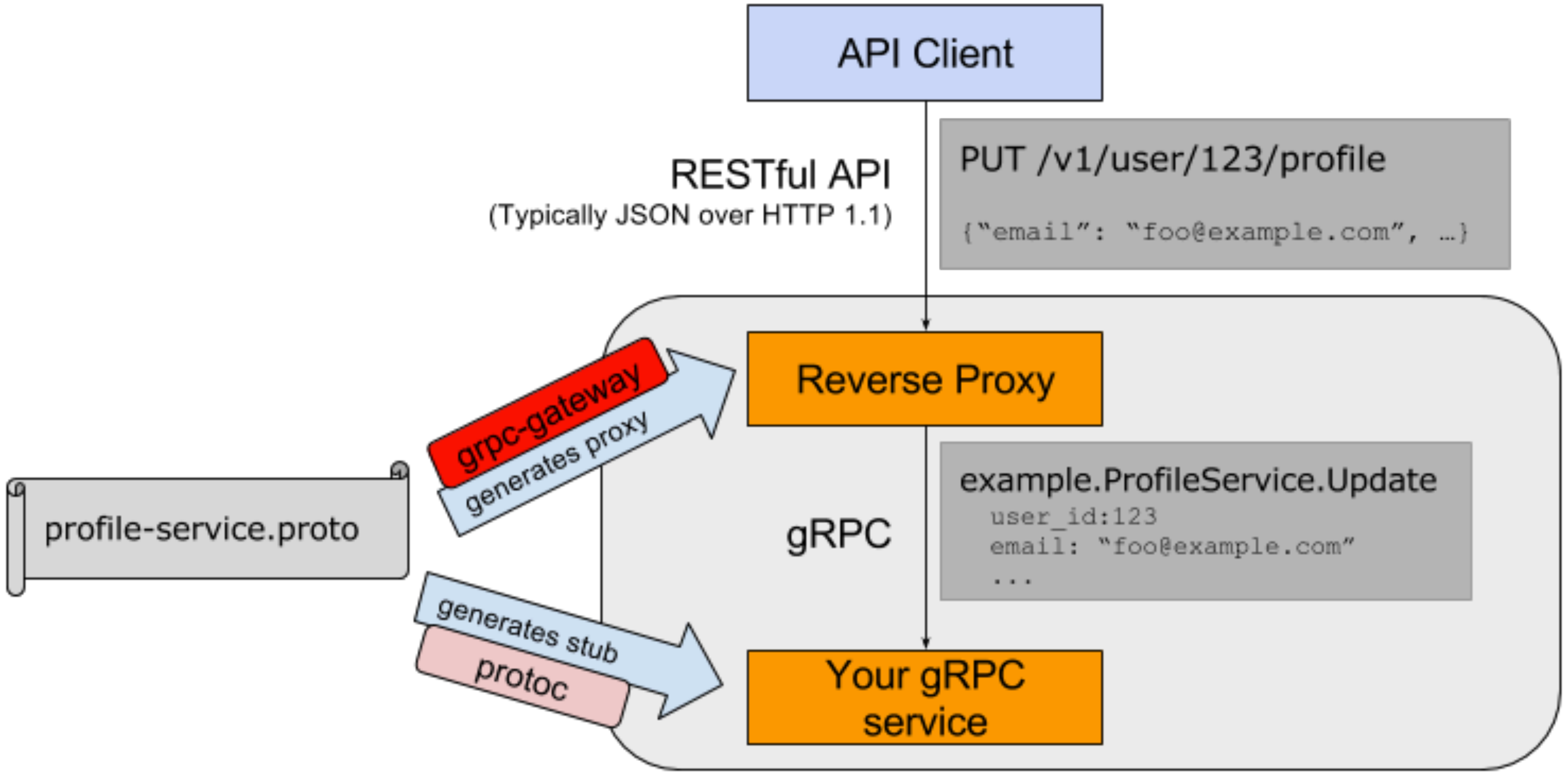
- Immediately terminates the RPC

- It is not a roll-back

Cancellation

- Can be initiated by both client/server
- Immediately terminates the RPC
- It is not a roll-back
- Automatically cascaded

Backward compatibility



API Client

RESTful API
(Typically JSON over HTTP 1.1)

```
PUT /v1/user/123/profile

{“email”: “foo@example.com”, …}
```

Reverse Proxy

gRPC

```
example.ProfileService.Update
  user_id:123
  email: “foo@example.com”
  ...
```

Your gRPC service

profile-service.proto

grpc-gateway
generates proxy

generates stub
protoc

https://github.com/grpc-ecosystem/grpc-gateway

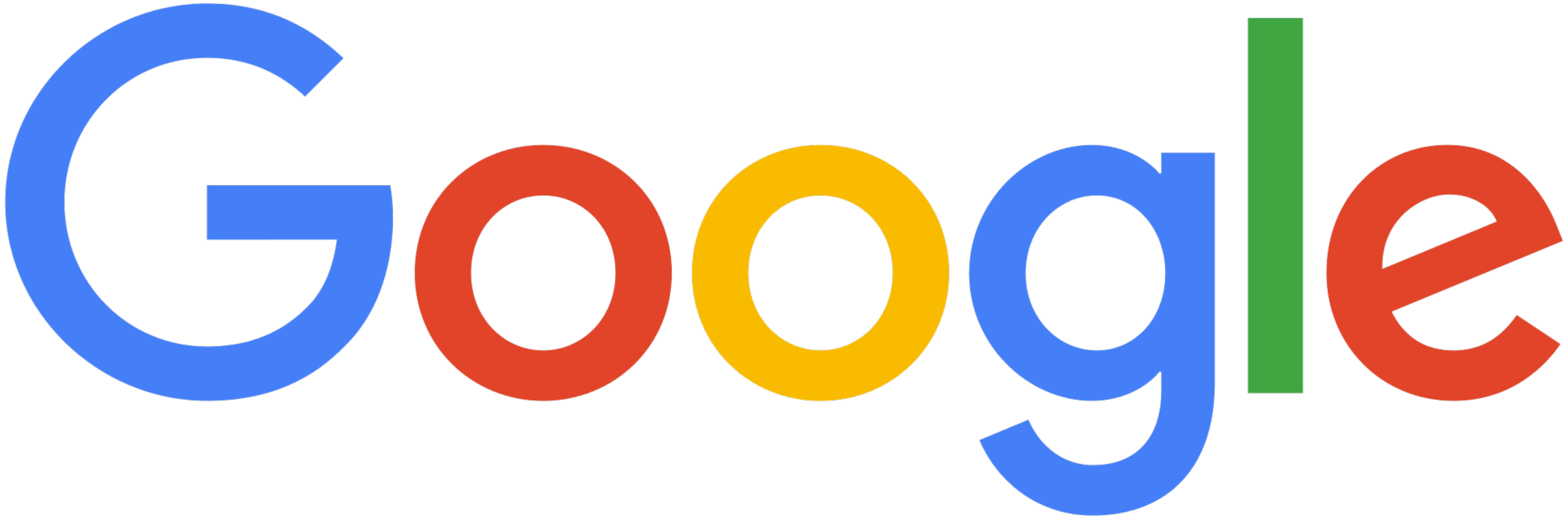
gRPC language support

- C++
- Python
- Java
- Go
- Ruby
- C#
- JS (Node)
- Android Java
- Objective-C
- PHP

gRPC Platform support

- Linux
- macOS
- Windows
- Android
- iOS

Success stories



Success stories

docker
Square
NETFLIX
CoreOS
cisco
carbon3D™
JUNIPer®
NETWORKS

Benefits

Benefits

–Focus on the API design & contract

Benefits

–Focus on the API design & contract

–HTTP2 is awesome

Benefits

–Focus on the API design & contract

–HTTP2 is awesome

–Bi-directional streaming

Benefits

–Focus on the API design & contract

–HTTP2 is awesome

–Bi-directional streaming

–Freedom to pick any language

Benefits

- Focus on the API design & contract
- HTTP2 is awesome
- Bi-directional streaming
- Freedom to pick any language
- Service-to-Service and Service-to-Mobile friendly

Benefits

–Focus on the API design & contract

–HTTP2 is awesome

–Bi-directional streaming

–Freedom to pick any language

–Service-to-Service and Service-to-Mobile friendly

–Production ready

Fin.

Thank **you**.